Yaohan Chu

Digitális számítógépek szervezése és a mikroprogramozás

Műszaki Könyvkiadó, 1977

Az eredeti mű címe:

Yaohan Chu: Computer Organization and Microprograming,
Copyright, © by Prentice-Hall, Inc., Englewood Cliffs,
New Jersey, USA

Fordította:

Szentgyörgyi Zsuzsanna
okl. villamosmérnök

Lektorálta:

Dr. Bohus Miklós
okl. villamosmérnök

Hungarian Translation

© Műszaki Könyvkiadó, 1977

ETO: 681.3.06—181.48
681.32

ISBN: 963 10 1845 8

Felelős kiadó: Solt Sándor igazgató

Felelős szerkesztő: Kemény Tamásné okl. villamosmérnök

Tartalomjegyzék

Előszó 13

1. A CDL tervező nyelv 15

1.1. Deklarációs utasítások 15
1.1.1. Regiszterutasítások 16
1.1.2. Tárutasítások 17
1.1.3. Kapcsolásokra vonatkozó utasítások 18
1.1.4. Indikátorutasítások 18
1.1.5. Terminál- (kapocspár-) utasítások 18
1.1.6. Az órajelgenerátor deklarálása 19
1.2. Mikroutasítások 20
1.2.1. Alapoperátorok 20
1.2.2. Kifejezések 21
1.2.3. Mikroműveletek 22
1.2.4. Feltételes mikroutasítások 23
1.2.5. Speciális operátorok 24
1.3. Sorrendbe állítás 25
1.3.1. Végrehajtási utasítás 25
1.3.2. Sorrendbe állítás többfázisú órajelgenerátor felhasználásával 26
1.3.3. Sorrendbe állítás fázisregiszterrel 27
1.3.4. Sorrendbe állítás többszörösen vezérelt sorozattal 27
1.4. Tárolt programú számítógép 28
1.4.1. Struktúra 29
1.4.2. Utasítás- és adatábrázolás 31
1.4.3. A gépiutasítás-készlet 31
1.4.4. Folyamatábra 32
1.4.5. Utasításokkal végzett leírás 34
1.4.6. Adat- és vezérlési utak 36
1.4.7. Példa speciális operátorok használatára 36
Feladatok 39
Irodalom 40

2. CDL alkalmazása vezérlési feladatok szimbolikus leírására 42

2.1. Soros paritásgenerátor 42
2.1.1. Soros paritásbit generálása 42
2.1.2. Logikai kapcsolási rajz 43
2.1.3. Folyamatábra 44
2.1.4. A műveletsorozat leírása 45
2.2. Soros komparátorok 46
2.2.1. Két bináris szám soros összehasonlítása 46
2.2.2. Logikai hatásvázlat előjel nélküli bináris számok összehasonlításához 47
2.2.3. Előjel nélküli bináris számok összehasonlításának folyamatábrája 47
2.2.4. Két, előjel nélküli bináris szám összehasonlításának szimulációja 50
2.2.5. Előjellel és abszolút értékkel ábrázolt bináris számok soros komparálása 50
2.3. A legnagyobb szám meghatározása 52
2.3.1. Algoritmus a legnagyobb szám meghatározására 52
2.3.2. A változat 52
2.3.3. B változat 56
2.4. Prímszámgenerátor 59
2.4.1. Prímszámok generálása 59
2.4.2. Software 60
2.4.3. Hardware 61
2.5. Gray—bináris kódkonverter 65
2.5.1. Gray—bináris kódkonverzió 65
2.5.2. Logikai hatásvázlat 67
2.5.3. Folyamatábra 68
2.5.4. A sorrendi működés leírása 69
2.6. Bináris—decimális konverter 69
2.6.1. Bináris—decimális konverzió 69
2.6.2. Logikai hatásvázlatok 70
2.6.3. Konverziós folyamat 71
2.6.4. A műveletek sorrendjének leírása 73
2.7. Tárolt átvitelű összeadás 74
2.7.1. A Mercer-összeadási algoritmus 74
2.7.2. Utasításleírás 75
2.7.3. Nullára ellenőrzés leírása 77
2.7.4. Az összehasonlítási teszt leírása 78
2.8. Tekeeredmény-számláló számítógép 78
2.8.1. A tekézés játékszabályai 79
2.8.2. Tekeponteredmény számítása 80
2.8.3. Logikai hatásvázlat 81
2.8.4. Folyamatábrák 83
2.8.5. Az utasítások sorrendjének leírása 86
Feladatok 87
Irodalom 87

3. Mikroprogramozás 89

3.1. Paritásgenerátor 89
3.1.1. Mikroprogramozott vezérlés logikai hatásvázlata 90
3.1.2. Mikroutasítás formátuma 91
3.1.3. A vezérlés sorrendjének meghatározása 91
3.1.4. Mikroprogram 93

3.1.5. Vezérlési ciklus 93
3.1.6. Összehasonlítás 95
3.2. Mikroprogramozott számítógép 95
3.2.1. Logikai hatásvázlat 95
3.2.2. Vezérlőjelek 97
3.2.3. A mikroutasítás formátuma 98
3.2.4. A műveletek sorrendjének leírása 98
3.2.5. Mikroprogram 100
3.3. Tárult logikájú számítógép 102
3.3.1. Logikai hatásvázlat 103
3.3.2. A műveletek sorrendjének leírása 104
3.3.3. Vezérlőjelek 105
3.3.4. A vezérlőszó formátuma 105
3.3.5. A műveletek sorrendjének leírása 106
3.3.6. Mikroprogram 108
3.4. Mikroprogramozott vezérlés egy sorozatból a legnagyobb szám meghatározására 108
3.4.1. Mikroprogramozott vezérlés logikai hatásvázlata 108
3.4.2. Időzítés és a vezérlőjelek előállítása 109
3.4.3. A mikroutasítás formátuma 111
3.4.4. A műveletek mikroprogramozása 112
3.4.5. Mikroprogram 114
Feladatok 114
Irodalom 115

4. Fixpontos aritmetikai egység 117

4.1. Az aritmetikai egység struktúrája 117
4.1.1. Fixpontos szám ábrázolása 118
4.1.2. Struktúra 118
4.1.3. Párhuzamos-összeadó 120
4.1.4. A Z terminálok 121
4.1.5. Az add2 operátor 121
4.2. Aritmetikai algoritmusok 121
4.2.1. Fixpontos összeadás és kivonás 122
4.2.2. Fixpontos szorzás 124
4.2.3. Fixpontos osztás 129
4.3. CDL leírások 131
4.3.1. Időzítő- és vezérlőjelek 132
4.3.2. Összeadási és kivonási sorozat 135
4.3.3. Szorzási sorozat 136
4.3.4. Osztási sorozat 137
4.4. Párhuzamos összeadó csoport- és szekcióátvitellel 138
4.4.1. Egybites teljes összeadó 138
4.4.2. A párhuzamos összeadó szervezése 139
4.4.3. Terminálutasítások a párhuzamos összeadók leírására 142
4.4.4. Csoportátvitelek 144
4.4.5. Szekcióátvitelek 147
4.5. Tervezési szempontok 148
4.5.1. Kétlépcsős regiszterek 148
4.5.2. Fixpontos aritmetikai utasítás 149
4.6. Az aritmetikai egység mikroprogramozása 150
4.6.1. Mikroprogramozott vezérlés struktúrája 151

4.6.2. Ütemezés és vezérlőjelek 153
4.6.3. Mikroutasítás-formátum 156
4.6.4. Az aritmetikai sorozatok mikroprogramozása 156
4.6.5. Mikroprogram 160
Feladatok 160
Irodalom 162

5. Párhuzamos-, bináris-, lebegőpontos aritmetikai egység 163

5.1. Az aritmetikai egység struktúrája 163
5.1.1. Lebegőpontos számábrázolás 164
5.1.2. A karakterisztika 165
5.1.3. Törtrész 165
5.1.4. Struktúra 166
5.2. Lebegőpontos összeadás és kivonás 168
5.2.1. Kezdetiérték-adás 169
5.2.2. Karakterisztikák azonos értékűvététele 170
5.2.3. Törtrész-összeadás és -kivonás 172
5.2.4. Normalizálás 172
5.3. Lebegőpontos szorzás 175
5.3.1. Kezdetiérték-adás 177
5.3.2. Karekterisztika-összeadás 177
5.3.3. Törtrészek szorzása 177
5.3.4. Normalizálás 178
5.4. Lebegőpontos osztás 179
5.4.1. Kezdetiérték-adás 180
5.4.2. Az osztandó törtrészeinek módosítása 181
5.4.3. Karakterisztika-kivonás 182
5.4.4. Törtrészek osztása 184
5.5. CDL leírás 184
5.5.1. Időzítő- és vezérlőjelek 186
5.5.2. Lebegőpontos összeadás és kivonás 187
5.5.3. Lebegőpontos szorzás 190
5.5.4. Lebegőpontos osztás 191
Feladatok 193
Irodalom 194

6. Soros aritmetikai egységek 195

6.1. Bináris, soros aritmetikai egység struktúrája 195
6.1.1. Fixpontos számábrázolás 196
6.1.2. Teljes összeadó—kivonó 196
6.1.3. Struktúra 197
6.2. Bináris összeadás és kivonás 200
6.2.1. Algoritmus 200
6.2.2. Túlcsordulási feltételek 201
6.2.3. Struktúra 201
6.2.4. Folyamatábra 203
6.3. Bináris szorzás 205
6.3.1. Algoritmus 205
6.3.2. Struktúra 205
6.3.3. Folyamatábra 206

6.4. Bináris osztás 208
6.4.1. Algoritmus 208
6.4.2. Osztás-stop állapot 208
6.4.3. Struktúra 210
6.4.4. Folyamatábra 212
6.5. Utasításleírások 213
6.5.1. Vezérlési struktúra 214
6.5.2. Összeadási és kivonási műveletsorozatok 215
6.5.3. Szorzási sorozat 216
6.5.4. Osztási sorozat 216
6.6. Decimális aritmetikai egység 217
6.6.1. Binárisan kódolt decimális számok 217
6.6.2. Működésmódok 219
6.6.3. Decimális összeadók és kivonók 220
6.6.4. Decimális, soros üzemű aritmetikai egység 224
6.6.5. Decimális összeadás és kivonás 226
6.6.6. Decimális szorzás 229
6.6.7. Decimális osztási algoritmus 230
6.7. Decimális szorzási és osztási eljárások 233
6.7.1. A szorzandó 1...9-szeresét használó szorzási eljárás 233
6.7.2. A kétszerező—felező módszert alkalmazó szorzóberendezés 234
6.7.3. Beépített szorzótáblát alkalmazó szorzóberendezés 235
6.7.4. Az osztó 9-szeresét felhasználó osztóberendezés 237
Feladatok 237
Irodalom 239

7. Társzervezés 240

7.1. Közvetlen elérésű tár 240
7.1.1. Tömbszervezés 240
7.1.2. Modulszervezés 242
7.1.3. Crossbarkapcsoló 244
7.1.4. Többszörös elérésű szervezés 245
7.1.5. A közvetlen elérésű tárak típusai 245
7.2. Tárcímzés 247
7.2.1. Közvetlen, direkt és indirekt címzés 248
7.2.2. Indexelt címzés 249
7.2.3. Relatív címzés 251
7.2.4. Báziscímzés 251
7.2.5. Regisztercímzés 251
7.3. Stack- (verem-) tárak 252
7.3.1. Stack (verem) szervezése 253
7.3.2. Stack- (verem-) műveletek 254
7.3.3. Stack- (verem-) beállítás 254
7.4. Asszociatív tár 255
7.4.1. Társzervezés 256
7.4.2. Egyeztetés numerikus kérdezőszóra 258
7.4.3. Egyeztetés logikai (Boole-féle) kérdezőszóra 265
7.4.4. Egyeztetés számlálási értékre 265
7.5. Dinamikus betöltő 266
7.5.1. A betöltő szervezése 266
7.5.2. Allokáció és betöltés 271
7.5.3. Utasítássorrend 275

7.5.4. Operanduscím lehívása 275
7.5.5. Elágazás, indexelés és indirekt címzés 276
7.5.6. Programvisszatérés és tárfelszabadítás 276
7.6. Puffertár szervezése 276
7.6.1. Puffertár 277
7.6.2. Puffertár szervezése 278
7.6.3. Puffertár-elérési sorozat 284
7.6.4. Teljesítőképesség értékelés 288
7.7. Vizuális tár 290
7.7.1. Alapfogalmak 291
7.7.2. Lapfelosztás 293
7.7.3. Szegmentálás 295
7.7.4. Szegmentált lapozás 297
7.7.5. Ütemezés 299
Feladatok 302
Irodalom 303

8. Vezérlés szervezése 306

8.1. Vezérlésszervezés sorrendi logikai hálózattal 306
8.1.1. Egylépéses vezérlés 307
8.1.2. Vezérlősorozat 307
8.1.3. Többszörös vezérlőjelek előállítása 308
8.1.4. Időzítés 309
8.2. Mikroprogramozott vezérlésszervezés 311
8.2.1. Mikroprogramozott vezérlőegység 311
8.2.2. Időzítés 313
8.2.3. A vezérlés hierarchiája 314
8.2.4. A mikroutasítás formátuma 315
8.2.5. Mikroprogramozott CPU 317
8.2.6. Mikroprogramozott B/K (I/O) vezérlőegység 324
8.3. A központi vezérlés szervezése 327
8.3.1. Sorrendbe állítás 328
8.3.2. Címzés 330
8.3.3. Prioritásos megszakítás 332
8.3.4. Megszakítási műveletsorozat 336
8.4. Aszinkron vezérlésszervezés 340
8.4.1. Tárolóegység 341
8.4.2. Be-, ill. kiviteli egység 344
8.4.3. Központi feldolgozóegység 347
8.4.4. Illesztések (interface) 350
8.5. A számítógéprendszer konfigurációja 352
8.5.1. Csatorna 352
8.5.2. Vezérlőegység 357
8.5.3. Moduláris konfiguráció 359
Feladatok 360
Irodalom 363

9. Számítógép-szervezés 364

9.1. Rendszertechnikai egységek 364
9.1.1. Főtár 364
9.1.2. Központi feldolgozóegység 366

9.1.3. Csatornák 366
9.1.4. Be-, ill. kimeneti készülékek és vezérlőegységek 367
9.1.5. Működtetőrendszer 367
9.2. Formátumok és kódok 368
9.2.1. Adatformátumok 368
9.2.2. Adatábrázolások 370
9.2.3. Adatkódok 371
9.2.4. Utasításformátumok 375
9.3. Számítógép-szervezés 377
9.3.1. Adatátvitel 377
9.3.2. Tárak 378
9.3.3. Regiszterek 382
9.3.4. A statok 384
9.3.5. Sínek 386
9.3.6. Csatornák 386
9.3.7. I/O interface (B/K illesztés) 387
9.3.8. Be-, ill. kimeneti vezérlőegységek és készülékek 388
9.3.9. Csatorna—csatornaadapter 388
9.4. Feldolgozóegység 389
9.4.1. Utasításkészlet 389
9.4.2. Általános rendeltetésű regiszterek 390
9.4.3. Feldolgozási üzemmódok 391
9.4.4. Feldolgozási műveletek 393
9.4.5. Kódátalakítás 393
9.5. A központi vezérlőegység 395
9.5.1. Az utasítások sorrendje 395
9.5.2. CPU állapot 396
9.5.3. Programállapotszó (Program Status Word) 397
9.5.4. Megszakítás 399
9.5.5. Mikroprogram 403
9.5.6. Megszakítás-felügyelőprogram 403
9.6. A felügyelőprogram és egyéb vezérlési módok 404
9.6.1. Tárvédelem 404
9.6.2. Időintervallum-adó (timer) 405
9.6.3. Várakozási állapot 406
9.6.4. Közvetlen vezérlés 406
9.6.5. Kezdeti-program betöltés 406
Feladatok 407
Irodalom 407

10. Csatornaszervezés 408

10.1. A be-, ill. kimenet vezérlésének szervezése 408
10.1.1. Direkt programozott vezérlés 409
10.1.2. Puffertárak alkalmazása 411
10.1.3. Adatcsatorna 412
10.1.4. Multiplexcsatorna 414
10.1.5. A be-, ill. kimeneti processzorok 416
10.2. A csatorna működése 418
10.2.1. A csatornacímszó 418
10.2.2. A csatornaparancsszó 418
10.2.3. A csatornaparancsok 420
10.2.4. A csatornaprogram 420

10.2.5. A csatornaállapot-szó 421
10.2.6. Az egységvezérlő szavak 423
10.2.7. A programállapotszó 423
10.2.8. Be-, ill. kimeneti (B/K) utasítások 424
10.2.9. Be-, ill. kimeneti (B/K) megszakítások 425
10.3. Be-, ill. kimeneti (B/K) interface 426
10.3.1. A be-, ill. kimeneti interface multiplexrendszerben 428
10.3.2. Illesztősín 428
10.3.3. Vezérlések ütemezése 431
10.3.4. Cím-, parancs-, állapot- és érzékelésbyte-ok 435
10.3.5. Interface-sorozatok 436
10.4. Szelektorcsatorna 440
10.4.1. Egységvezérlő szó 440
10.4.2. Adatáram 441
10.4.3. Adatátvitel (szolgálatkérés) 443
10.4.4. Be-, ill. kimeneti (I/O) start 445
10.4.5. Egyéb be-, ill. kimeneti utasítások 448
10.5. Multiplexcsatorna 448
10.5.1. Csatornavezérlő szó 450
10.5.2. Csatornatesztelés 451
10.5.3. Be-, ill. kimenet állj 451
10.5.4. Be-, ill. kimenet tesztelése 451
10.5.5. Be-, ill. kimeneti start 454
Irodalom 457

11. A software mikroprogramozott implementálása 458

11.1. Áthelyezhető kód végrehajtható kóddá való fordítása 458
11.1.1. Áthelyezhető elemek 459
11.1.2. Végrehajtható kód 463
11.1.3. Fordítási algoritmus 464
11.2. Struktúra 467
11.3. Műveletsorozatok 469
11.3.1. Iniciálási sorozat 473
11.3.2. Lehívási sorozat 473
11.3.3. Szétválasztási sorozat 474
11.3.4. Címmódosítási sorozat 474
11.4. Mikroprogramozott vezérlés 475
11.4.1. Vezérlési struktúra 475
11.4.2. Időzítő- és vezérlőjelek 475
11.4.3. Mikroutasítás-formátum 477
11.5. Mikroprogramozás 479
11.5.1. Szétválasztási sorozat 480
11.5.2. Iniciálási sorozat 481
11.5.3. Lehívási sorozat 482
11.5.4. Címmódosítási sorozat 483
11.5.5. Mikroprogram 486
Feladatok 487
Irodalom 487
A könyvben gyakrabban használt angol kifejezések magyar megfelelője 489
Tárgymutató 491

Előszó

A számítógép belső szervezése határozza meg a digitális számítógép funkcionális működését. Könyvünk célja e belső szervezésnek az ismertetése, és ezért nem foglalkozik sem az elektronikus áramkörökkel, áramköri modulokkal, szerelvényekkel, belső huzalozással, tehát a digitális számítógép tényleges felépítésével, sem pedig a kapuk, flip-flopok, kapcsolók, kijelzők, tárak — vagyis a számítógépet alkotó főbb elemek — közötti összefüggések leírásával. Ez a könyv a digitális számítógépek adatformátumait, számábrázolásait, utasításkészletét, konfigurációit, a bennük végbemenő folyamatok sorrendjét, ezek időzítését, a különböző parancsokat és vezérléseket ismerteti. Röviden: a számítógép belső szervezését, tehát a digitális számítógép utasításainak (funkcióinak) szervezését és sorrendi működését írja le.

A számítógép belső szervezését rendszerint szóbeli magyarázatokkal ellátott tömbvázlatokkal, blokkdiagramokkal írják le. Ez azonban csak az általános működés megértéséhez ad segítséget. Szükséges azonban a belső szervezés részletesebb leírása is. Ehhez valamilyen magas szintű nyelvet kell alkalmazni. E könyvben a számítógépet felépítő elemeknek és a gép sorrendi műveleteinek leírására a CDL (**C**omputer **D**esign **L**anguage) nyelvet használjuk. A nyelv tömör és pontos leírást szolgáltat. Az algoritmusok leírására folyamatábrákat is felhasznál. A folyamatábra segítségével leírhatók a párhuzamos műveletek, a szimultán folyamatok és a független hurkok. A tárgyalás során a nyelv mindkét változatát — a procedurális és a nemprocedurális változatot is — használjuk. A könyv részletesen bemutatja a CDL használhatóságát a hardware-ben implementált komplex algoritmusok leírására.

Készült olyan CDL szimulátor, amely a CDL részhalmazát tartalmazza. Négy működő változata van: az IBM 7094-re, az Univac 1108-ra, a CDC 6600-ra és az IBM S/360 számítógéprendszerekre. Az Univac 1108 változat adagolt (batch) és terminálról jövő feldolgozást is lehetővé tesz. Az első változatot a Marylandi Egyetem Számítástudományi Központjában (Computer Science Center of University of Maryland) dolgozták ki. A CDC 6600 változatot a Naval Ordinance Laboratory készítette. Az IBM S/360 változatot a Torontói Egyetem Számítógép Rendszerek Csoportja (Computer Systems Group at the University of Toronto) dolgozta ki.

A könyv tizenegy fejezetre tagolódik. Az 1. fejezet bevezet a CDL-be. A 2. fejezet a CDL alkalmazását illusztrálja, néhány egyszerű szervezési példa leírásával (komparátor, kódkonverter és egy labdajáték ütéseit számláló számítógép). A 3. fejezet a mikroprogramozott számítógép belső működését és a mikroprogramozás elvi vonatkozásait ismerteti; leírásra kerül egy egyszerű mikroprogramozott számítógép és egy

tárolt logikájú számítógép. A mikroprogramozásról szól a 4., 8. és 11. fejezet is. A 4. fejezet egy párhuzamos, bináris, fixpontos aritmetikai egységet mutat be, huzalozott és mikroprogramozott vezérléssel, míg az 5. fejezet egy párhuzamos, bináris, lebegőpontos aritmetikai egységet ír le. A 4. fejezet a párhuzamos összeadó működését mutatja be, hasonlót ahhoz, amilyen az IBM 7090 számítógépcsaládban található. A 6. fejezet soros üzemű aritmetikai egységeket szimulál bináris és decimálisan kódolt bináris számábrázolással. A 7. fejezet a véletlen hozzáférésű tárat, az asszociatív tárat, a virtuális tárat, tárcímzési elveket, a stack-et (veremtárat) és a tárbetöltés funkcionális szervezését írja le. Bemutatásra kerül az IBM 360-as rendszer 85-ös modelljének társzervezéséhez hasonló szervezési elv. A 8. fejezet különböző vezérlések szervezésével foglalkozik, tárgyalja a huzalozott, a mikroprogramozott, az aszinkron és a központi vezérlésszervezés elveit. A 9. fejezetben a számítógép belső szervezésének főbb egységei kerülnek bemutatásra és illusztrálásra, ebből megismerhető egy teljes számítógép belső szervezése. A választott géptípus az IBM S/360 számítógépcsalád közepes kategóriájú, 40-es modellje. A 40-es modell csatornaszervezését a 10. fejezet írja le. A software-nek mikroprogramozással való megvalósítását a 11. fejezetben példán mutatjuk be. A példa egy áthelyezhető program végrehajtható programmá való fordításának mikroprogramozását mutatja be.

A szerző köszönetét szeretné kifejezni az ACM-nek, az IEEE-nek, az AFIPS-nak, valamint a Computer Design című lapnak és különösen az IBM-nek, hogy engedélyt adtak különböző publikációk felhasználására. Ugyancsak elismerését kívánja kifejezni azoknak az egyetemi hallgatóknak, akik segítségére voltak a könyv elkészítésében, különösen O. R. Pardon-nak a 11. fejezetben végzett munkájáért és Nancy Nowell-nek, aki az egész kézirat gépelését végezte.

Chevy Chase, Maryland

Yaohan Chu

1.
A CDL tervezőnyelv

A számítógép belső szervezését rendszerint magyarázó szöveggel ellátott tömbvázlatokkal írják le. Az ilyen leírási mód ugyan általános áttekintést ad a szervezésről, de hiányzik belőle az a részletesség és az a mélység, ami lehetővé teszi az alapos megértést. Ezért kívánatos valamilyen magas szintű nyelv kifejlesztése a számítógép belső szervezésének leírására.

Az elmúlt években számos ilyen magas szintű nyelvről jelent meg publikáció [2]...[18]. Közéjük tartozik a Computer Design Language vagy egyszerűen CDL [7], amelyet digitális számítógépek funkcionális szervezésének, algoritmikus működésének és szekvenciális műveleteinek leírására fejlesztettek ki. A nyelv erősen leíró jellegű, amikor olyan számítógépelemeket indentifikál, mint amilyenek pl. a regiszterek, dekódolók, kapcsolók, kijelzők, tárak és terminálok. Az elemek, algoritmusok és műveletek leírásában pontos; és nemcsak bitszinten, szószinten és bitsorozatszinten igen kifejező, hanem az időzítőjelek és vezérlőparancsok szintjén is. A CDL soros és párhuzamos átviteleket és párhuzamos műveleteket is le tud írni. Nagy előnye, hogy a nyelv megértéséhez és használatához nincs szükség elektronikai ismeretekre.

Ebben az első fejezetben bemutatjuk a CDL-t, amelyet a könyvben következetesen használunk, valamint ismertetjük az egyszerű tárolt programú digitális számítógép funkcionális elemeinek, mikroműveleteinek, ezek sorrendjének, és végül a teljes gépnek a leírását.

1.1.
Deklarációs utasítások

Az általunk tárgyalt digitális számítógép tárolt programú gép, amely hat regisztert, egy véletlen hozzáférésű tárat (RAM), három kapcsolót, egy kijelzőt és egy órajelgenerátort tartalmaz. A számítógép felsorolt funkcionális elemeit deklarációs utasítások[1] írják le, amelyek nemcsak azonosítják az elemeket, hanem szimbolikus nevet is rendelnek minden egyes elemhez, s ha szükséges, specifikálják azok funkcióját is. Ezeket a funkcionális elemeket a következőkben ismertetjük.

[1] Mivel a magyar szóhasználatban a „statement" is, az „instruction" is utasítás, de az első általában nyelvi utasítást, a második pedig gépi utasítást jelent, megkülönböztetésként a továbbiakban a *statementet utasításnak* és az *instructiont gépi utasításnak* nevezzük. Megjegyezzük, hogy ez a két utasításszint a legújabb fejlesztésű számítógépekben egybe is eshet. *(Fordító)*

1.1.1
Regiszterutasítások

Az ismertetésre kerülő tárolt programú számítógép hat regisztere a következő:

R(0—23), A(0—23), C(0—14), D(0—14), F(0—5) és G.

Mind az R, mind az A regiszter 24 bitből áll. Ez a regiszterekhez tartozó zárójelben levő számból, az „index"-ből állapítható meg. Balról a legelső bit esetünkben az R(0) vagy A(0) és jobbról a legutolsó bit az R(23) vagy A(23). Az R regiszter a főtár tartalom (adat-) regisztere, míg az A regiszter a gép akkumulátora, amelyben az összeadás, kivonás eredménye tárolódik és amelyben a tárolt operandus léptethető. A C és a D regiszter egyaránt 15 bitből áll. Az előbbi a tár címregisztere, míg az utóbbi az utasításszámláló (vagy a következő utasítás címének regisztere). A G regiszternek — mivel példánk szerint egybites — nincs indexe. Ez vezérli a számítógép működését vagy leállását. A regisztereket *regiszterutasítások* specifikálják. Ilyen a felsorolt regisztereket specifikáló utasítás pl. a következő:

```
Register,        R(0—23), A(0—23), C(0—14), D(0—14), F(0—5), G        (1.1)
```

A jobb olvashatóság kedvéért megjegyzéseket[2] (*comment-teket*) lehet a regiszter utasításba iktatni. Például az (1.1) utasítás a következőképpen írható át:

```
Register,   R(0—23),   $tartalom- (adat-) regiszter
            A(0—23),   $akkumulátor
            C(0—14),   $címregiszter                                  (1.2)
            D(0—14),   $utasításszámláló (regiszter)
            F(0—5),    $fázisregiszter
            G,         $start—stop (vezérlő) regiszter
```

A $ (dollár) jellel kezdődő kifejezések megjegyzések.

A részregiszter csupán része a regiszternek, de általában önálló jelentéssel rendelkező biteket tartalmaz. Ezeknek a részeknek szimbolikusan indexelt nevet lehet adni egy *subregister-* (*részregiszter-*) utasítással. Például a következő regiszter utasítás a

```
Register,     R(0—23),  részregiszterei:
Subregister,  R(OP)=R(0—5), R(CIM)=R(9—23)
```

Ez azt deklarálja, hogy az R regiszter bal oldalról számított legelső hat bitje a szimbolikusan indexezett R(OP) nevet, a jobb oldali 15 bitje pedig a szimbolikusan indexezett R(CIM) nevet kapja. Az R(OP) és R(CIM) részregiszterek sorrendben, egy utasítás operációs (műveleti) kód részét és címrészét tartalmazzák.

Néha két vagy több regisztert vagy részregisztert egyetlen regiszterbe egyesítenek, speciális funkciók megvalósítására. Ezt regisztersornak[3] vagy *casregister-nek* nevezik, amit viszont *casregister-utasítással* lehet leírni.

Nézzük pl. a következő casregister-utasítást:

```
Register,     A(1—5), PB,
Casregister,  D(1—6)=A—PB
```

Itt a D olyan *regisztersort* deklarál, amely az A regiszterhez kapcsolt PB regiszterből

[2] A deklarációs utasításokban szereplő deklarálók (pl. register, casregister, light, clock stb.) eredeti angol elnevezését meghagyjuk, de a magyarázó szövegben magyar megfelelőiket használjuk (pl. regiszter, regisztersor, indikátor, órajelgenerátor stb.). A deklarálók a nyelv részei, ezért azokat nincs jogunk megváltoztatni, ugyanúgy, ahogy pl. az ALGOL 60-ban is eredeti angol nyelvű deklarálókat (real, integer, array, stb.) használunk. Ugyanez vonatkozik a *commentre* is. *(Fordító)*

[3] A „cascaded register" rövidítése „casregister". Magyar megfelelője a sorbakapcsolt, felfűzött regiszter, ill. annak rövidítése, a regisztersor. *(Fordító)*

áll. Ez a regiszteregyesítési (összevonási) utasítás nem szükséges, ha nem kell új nevet bevezetni. Másik példaként nézzük a következő regiszteregyesítési utasítást:

```
Register,     A(0—23), Q(0—23),
Subregister,  Q(M)=Q(0—23)                    (1.3)
Casregister,  AQ(0—46)=A—Q(M)
```

Itt az AQ regisztersort deklarál, amely az A regiszterhez kapcsolt Q(M) részregiszterből áll. Megjegyezzük, hogy a regisztersor nem újonnan létrehozott regiszter, hanem egy olyan regiszter, amelyet a korábban deklarált regiszterekből fűztek fel, vagyis ezek kaszkád kapcsolódásából áll.

Szükségünk van arra is, hogy több azonos regisztert is le tudjunk írni, amelyek flip-flopokból (azaz egy bites regiszterekből) álló tömböt (mátrixot) alkotnak. Legyen példa erre a négy azonos, A1(0—8), A2(0—8), A3(0—8) és A4(0—8) regiszternek egyetlen nyolcszámjegyű decimális szám tárolására való alkalmazása. Ahelyett, hogy négy regiszterként deklarálnánk őket, kényelmesebb egyetlen regisztertömbként deklarálni, egyetlen *array-register* (regisztertömb-) *utasítással*:

```
Array-register, A(1—4, 0—8), Q(1—4, 1—8)      (1.4)
```

Az (1.4) utasítás az A-nak nevezett, flip-flopokból képzett tömböt ír le, amelynek négy sora és kilenc oszlopa van, míg a másik, itt leírt Q tömb négy sorból és nyolc oszlopból áll. Mindegyik 4-bites oszlop egy binárisan kódolt decimális számjegyet (l. a 6. fejezetet) tárol. A tömb egy-egy sorát vagy oszlopát úgy lehet specifikálni, hogy elhagyjuk az indexet, de meghagyjuk a vesszőt. Például az A tömb második sora az A(2,) részregiszterrel jelölhető, a Q tömb nyolcadik oszlopa pedig a Q(,8) részregiszterrel.

A regisztersor helyett regisztersorokból álló regisztertömböt (*array-casregister*) használhatunk. Például, az előbb említett A és Q regisztertömbök felfűzését (kaszkád kapcsolását) a következő regisztersorokból álló regisztertömb-utasítással lehet meghatározni:

```
Array-casregister,  AQ(1—4, 0—16)=AQ          (1.5)
```

A későbbiekben bemutatásra kerülő alapműveleteket (operátorokat) alkalmazni lehet array-registerekre és array-casregisterekre egyaránt.

1.1.2. Tárutasítások

Tárolt programú számítógéphez szükség van egy véletlen hozzáférésű tárra, amely nagy számú regiszterből — *tárhelyből* — épül fel. Minden egyes tárpozícióhoz egy *tárcím* rendelhető és a tár el van látva címregiszterrel és tartalom- (adat-) -regiszterrel. Ha a tár valamely helyéhez hozzá akarunk férni, a hely címét a címregiszterben kell elhelyezni, mielőtt a tárral kapcsolatos művelet végbemenne. Egy egyszerű véletlen hozzáférésű tárat egyetlen tár- (*memory*) *utasítással* lehet specifikálni. Az egyszerű számítógépünk véletlen hozzáférésű tárát leíró tárutasítás:

```
Register,   C(0—14),                $címregiszter,
Memory,     M(C)=M(0—32767, 0—23)                  (1.6)
```

Az (1.6) tárutasítás deklarálja az M (memóriát) tárat, a C pedig címregisztert specifikál oly módon, hogy az M(C) indexe a C. Az utasítás meghatározza, hogy a tár kapacitása 32 768 szó és egy szó hossza 24 bit. A fentiekből látható, hogy egy regiszter leírásához egyetlen indexre, míg a tár egyértelmű specifikálásához két indexre van szükség.

1.1.3.
Kapcsolókra vonatkozó utasítások

Kézi kapcsolókat rendszerint a digitális számítógép vezérlőpultján használnak azért, hogy a gépkezelő — „kívülről" — vezérelni tudja a számítógépet. A kapcsoló tehát olyan bemeneti eszköz, amelynek egy vagy több pozíciója van. Ha egyetlen pozíciója van, akkor elfordításakor egy jelet állít elő. Ha több pozíciója van, mindig abban az állásában marad, amelybe fordították, tehát jelzi (tárolja) az adott állapotot, vagyis egyúttal tárolóelem is.

A kapcsolókat *switch-* (*kapcsoló-*) *utasítással* lehet specifikálni. Legyen három kapcsoló az egyszerű digitális számítógépünkben. Ezeket a következő kapcsolóutasításokkal lehet leírni:

```
Switch,    POWER(ON),
           START(ON),                    (1.7)
           STOP(ON).
```

Az (1.7) utasítás három egyállású kapcsolót deklarál, amelyeknek ugyanaz az — ON (BE) — a pozíciónevük. A POWER (tápforrás) kapcsoló készíti elő a számítógép működését, míg a START és a STOP kapcsolók a számítógép működésének indítását, ill. leállítását vezérlik. A kapcsolóutasítás egyetleni ndexhalmazzal rendelkező kapcsolókat is deklarálhat, így kapcsolókból álló sor írható le, ill. két indexhalmazú kapcsolók alkalmazhatók kapcsolókból álló tömb leírására.

1.1.4.
Indikátorutasítások

Az indikátorok rendszerint a számítógép vezérlőpultján vannak azért, hogy a gépkezelő a számítógép állapotáról jelzést kapjon. Tehát az indikátor — kijelző — olyan kimeneti eszköz, amelynek egy vagy több állapota van. Ha csak egyetlen pozíciója van, a bekapcsolásakor felvillan. Ha két vagy több pozíciója van, abban a világító állapotban marad, amelybe bekapcsolták. Tehát az indikátor is tárolóelem.

Az indikátorokat *light- (indikátor-) utasítással* lehet specifikálni. Az alábbi utasítás a számítógépben bekövetkező túlcsordulást mutató kijelzőt írja le:

```
Light,  INDOV (ON, OFF)                  (1.8)
```

Az INDOV kijelzőnek két pozíciója van, az ON (BE) és az OFF (KI). Akkor kerül ON állásba, ha összeadáskor túlcsordulás keletkezik. Egy vagy két indexszel rendelkező indikátorokat deklaráló utasítás indikátorokból álló sort vagy tömböt tud leírni.

1.1.5.
Terminál- (kapocspár-) utasítások

Egy kombinációs logikai hálózat valamilyen speciális funkciójú, egymással összekapcsolt olyan logikai áramkörök csoportja, amelyek nem tartalmaznak tárolóelemet. Például a párhuzamos teljes összeadó, amely két, előjel nélküli bináris számot ad össze párhuzamosan, egy kombinációs logikai hálózat.

A logikai hálózatot *terminal- (kapocspár-) utasítás* írja le. Például egyszerű digitális számítógépünk párhuzamos összeadójának kimeneteit ez esetben az alábbi kapocspár-utasítás írja le:

```
Comment,    párhuzamos teljes összeadó leírása,                    (1.9)
Register,   A(0—23), R(0—23),
Terminal,   C(23)=0,
            C(0—22)=A(1—23)*R(1—23)+R(1—23)*C(1—23)+
                    +C(1—23)*A(1—23),
            SUM(0—23)=A(0—23)⊕R(0—23)⊕C(0—23).
```

A fenti utasításokban az A és R regiszter a párhuzamos összeadó két bemenete, és SUM a kimeneti kapocspár. A C(0—22) az átvitelkimenet a párhuzamos összeadó egyes helyértékein. A C(23) átvitelhez a 0 értéket rendeljük hozzá. Jegyezzük meg, hogy az (1.9) utasításban leírt párhuzamos összeadó összeadja az A és az R regiszterben levő két, előjel nélküli bináris számot, de nem jelzi, hogy ha összeadáskor túlcsordulás következik be.

A *dekódoló* egy másik példa a kombinációs logikai hálózatokra. A dekódoló valamely regiszter minden egyes értékéhez egy és csakis egy kimenetet rendel hozzá. Az *n* bitből álló regiszter esetén, a hozzá rendelt dekódolónak összesen 2^n kimenete lehet. Ezért igen kényelmetlen lenne egy nagy számú kimenettel rendelkező dekódoló minden kimenetét egy-egy utasítással leírni. Ezért a dekódolókat *decoder- (dekódoló-) utasítások* specifikálják. A következőkben egy olyan dekódolóutasításra láthatunk példát, amely egyszerű számítógépünk vezérlőegységében levő dekódolót írja le:

```
Register,   F(0—5),
Decoder,    K(0—16)=F                                              (1.10)
```

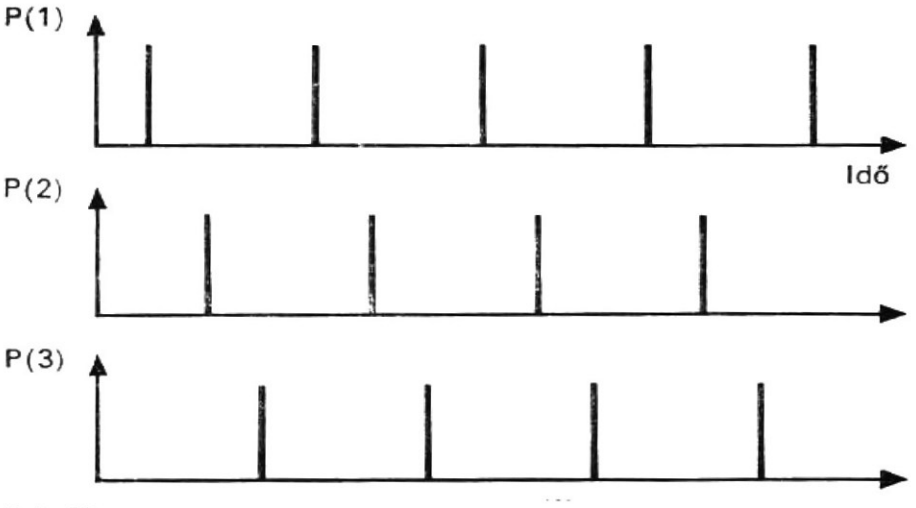
Ez a dekódolóutasítás azt deklarálja, hogy a K(0),...,K(16) az F (fázis-) regiszterhez rendelt dekódoló kimeneti kapocspárja. Minthogy az F regiszter hatbites, a dekódolónak összesen 64 kimeneti kapcsa lehet. Azonban a (0—16) index azt mutatja, hogy csak az első 17 kimenetre van példánkban szükség.

1.1.6. Az órajelgenerátor deklarálása

Az órajelgenerátor olyan elektronikus eszköz, amely impulzussorozatot vagy sorozatokat generál abból a célból, hogy a számítógépben levő logikai áramkörök digitális működését időzítse. Két szomszédos órajelimpulzus közötti időintervallumot az órajel periódusidejének nevezik. Az órát *clock-utasítás* (clock: óra) írja le. A számítógép háromfázisú órajelgenerátorát leíró utasítás a következő alakú:

```
Clock,      P(1—3)                                                 (1.11)
```

Az (1.11) utasítás azt deklarálja, hogy az óraimpulzusok neve P és a három fázis neve P(1), P(2) és P(3). A három fázisban az órajelimpulzusok periódusideje és késleltetési ideje egyforma, amint az az 1.1 ábrán is látható. Minden fázisban egy impulzus periódusidőt tartalmazó két szomszédos impulzus képez egy ciklust, ezt *órajelciklusnak* nevezik. Az (1.11) utasításban levő órajelgenerátor órajelciklusának három egymást követő impulzusát három lépésből álló sorozat vezérlésére lehet használni.



1.1. ábra
A háromfázisú órajelgenerátor impulzusai

1.2. Mikroutasítások

A mikroművelet olyan elemi, funkcionális művelet, amely fizikailag be van építve a digitális számítógép mikroprogramjába. A mikroműveletet mikroutasítás írja le. Az alábbiakban bemutatjuk a mikroutasításokat alkotó operátorokat, kifejezéseket és átviteleket.

1.2.1. Alapoperátorok

Az operátor olyan műveletet jelképező szimbólum, amely a logikai hálózat által egyetlen órajelperiódusban végrehajtandó funkciót reprezentál. A gyakran használt operátorokat *alapoperátoroknak* (bázisoperátoroknak) nevezik. Az ebben a könyvben felhasználásra kerülő alapoperátorok felsorolása az 1.1. táblázatban található. Az általuk végrehajtott funkciókat a következőkben magyarázzuk meg.

Az alapoperátorokat logikai és aritmetikai operátorokra bonthatjuk. A Boole-algebrából öt logikai operátort használunk, ezek: NEM, VAGY, ÉS, ANTV (antivalencia, kizáró VAGY) és EKV (ekvivalencia, azonosság). Jelölésük: NOT, OR, AND, EXOR, COIN. Ezeknek a logikai operátoroknak mindegyike vonatkozhat egyetlen bitre és egyedileg bittöbbesekre is. Hat regiszteroperátort értelmezünk. Ezek: shl, shr, cil, cir, countup vagy inc (előreszámlálás, inkrementálás), countdn vagy dec (visszaszámlálás, dekrementálás)[4]. A léptető- (shift-) operátorok — az shl és az shr — egy regiszter tartalmát egy vagy több bittel balra, ill. jobbra léptetik, miközben — megfelelően — egy vagy több nullát iktatnak be a regiszter jobb, ill. bal oldali végén. A ciklikus-léptetőoperátorok (cil, cir) szintén léptetőoperátorok, azzal a különbséggel, hogy a regiszter egyik vége össze van kötve a másik végével. A számlálóoperátorok (countup vagy inc, ill. countdn vagy dec) eggyel növelik, ill. csökkentik egy regiszter tartalmát. Van két aritmetikai operátor (az add és a sub), amelyek egy előjel nélküli bináris számot hozzáadnak, ill. kivonnak egy előjel nélküli bináris számból.

[4] A bázisoperátorokat ugyanazon megfontolások szerint, mint a deklarálókat (l. [2] lábjegyzet), eredeti nevükön használjuk *(Fordító)*

1.1. táblázat

Alap- (bázis-) operátorok

Operátor	Szimbólum	Magyarázat
*Logikai operátorok**		
NEM (NOT)	'	Logikai Nem művelet
VAGY (OR)	+	Logikai Vagy művelet
ÉS (AND)	*	Logikai És művelet
ANTV (EXOR)	⊕	Logikai Kizáró vagy művelet
EKV (COIN)	⊙	Logikai Azonosság művelet
Regiszteroperátorok		
Léptetés balra	Shl	Léptetés balra egy vagy több bittel**
Léptetés jobbra	Shr	Léptetés jobbra egy vagy több bittel**
Ciklikus léptetés balra	Cil	Ciklikus léptetés balra egy vagy több bittel
Ciklikus léptetés jobbra	Cir	Ciklikus léptetés jobbra egy vagy több bittel
Előreszámlálás	Countup, inc	Inkrementálás eggyel
Visszaszámlálás	Countdn, dec	Dekrementálás eggyel
Aritmetikai operátorok		
Összeadás	Add	Előjel nélküli bináris szám hozzáadása egy másikhoz
Kivonás	Sub	Előjel nélküli bináris szám kivonása egy másikból

* A logikai operátorok egyetlen bitre, vagy egyedenként, bittöbbesekre vonatkozhatnak.
** Nullákat kell beiktatni a jobb oldalra — balra léptetéskor, és a bal oldalra — jobbra léptetéskor. (A szerző jelölése nem tartalmazza több-bites léptetés esetén a lépésszámot.)

Egy operátor lehet unáris vagy bináris (monadikus vagy diádikus) vagy mindkettő. Az unáris operátorhoz egy, a binárishoz két operandus szükséges. A NOT (NEM), a countup és a countdn unáris operátorok. A +, *, ⊕, ⊙, add és sub operátorok binárisak. Az 1.1. táblázatban található többi operátor lehet unáris is, bináris is. Ezen operátorok alkalmazására a következő fejezetekben látunk példákat.

1.2.2. Kifejezések

A kifejezés konstans, operátor és/vagy operandus kombinációja, amely egy utasítás vagy mikroutasítás egy részének leírására szolgál. A kifejezés azt a funkciót reprezentálja, amelyet pl. a mikroműveletnek végre kell hajtania, mielőtt benne átvitel (transfer) következnék be. Egy kifejezés pl. a következőkből állhat: egyetlen operandusból (amilyen mondjuk, az A vagy az OV (**OV**erflow: túlcsordulás) unáris operátorból egyetlen operandussal (mint pl. shl A és countup D); vagy pedig bináris operátorból két operandussal (mint pl. A add R). A kifejezés lehet ezen kívül bináris, oktális vagy decimális konstans is. A konstansok indexelésére vonatkozó konvenciót vezessük be a következőképpen: *a decimális konstans nincs indexelve, az oktális konstans mindig*

indexelve van; a bináris konstans csak akkor van indexelve, ha nem nyilvánvaló, hogy bináris. Néhány példa kifejezésekre:

$$\begin{array}{l} X' \\ \text{shl Y(ADDR) vagy shl Y(CIM)} \\ X \oplus Y \\ 3 \text{ shl } X \\ 5 \text{ cir } Y \end{array} \qquad (1.12)$$

ahol X és Y regiszterek, míg az Y(ADDR) részregiszter.

1.2.3. Mikroműveletek

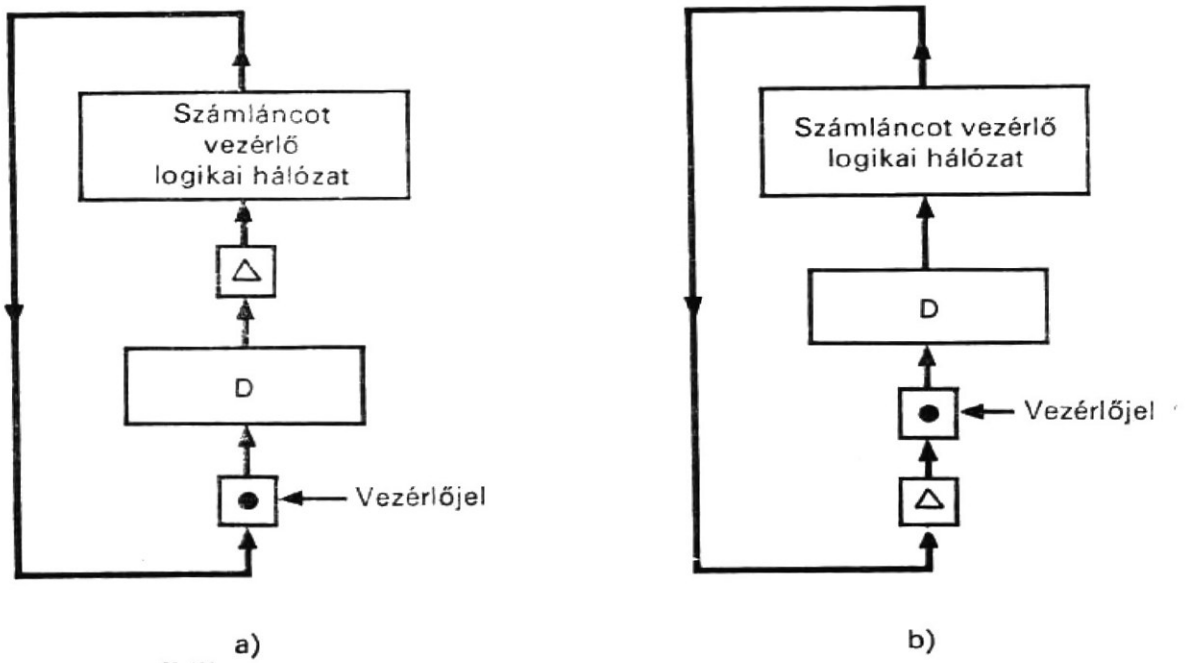
A mikroművelet egy konstanst vagy egy vagy két regiszter tartalmát viszi át egy másik regiszterbe; a művelet ilyenkor az átvitel alatt kerül végrehajtásra. Ez az egyik regiszterből a másikba való átvitel a mikroművelet jellemzője.

Amint az 1.2. szakaszban már említettük, a mikroműveletet egy mikroutasítás írja le, amelynek két része lehet: az egyik a mikroművelet operandusait és operátorát, míg a másik az átvitelt és a mikroművelet eredményét mutatja. Az első rész egy kifejezés, a második pedig egy nyílból (az átvitel jelölésére) és annak a regiszternek a nevéből áll, ahol az eredményt tárolni kell. Vegyünk egy olyan mikroműveletet, amely az R regiszter tartalmát átviszi az A regiszterbe. A mikroműveletet leíró mikroutasítás elölésünk szerint a következő alakú:

$$A \leftarrow R \qquad (1.13)$$

Jegyezzük meg, hogy ebben a mikroutasításban a nyíl szimbolizálja az átvitelt, és a kifejezés (a nyíl jobb oldalán) mindössze egy regiszter neve.

Fontos megjegyeznünk azt, hogy a mikroműveletben az átvitel fizikailag nem pillanatszerűen megy végbe, végrehajtásához véges időre van szükség. Rendszerint ennél



1.2. ábra
Késleltetések a regiszterátvitel útvonalán:
***a)* késleltetés a regiszter kimeneti oldalán; *b)* késleltetés a regiszter bemeneti oldalán**

az időnél nagyobb az órajelgenerátor periódusideje, hogy a mikroműveletet be lehessen fejezni egyetlen órajelperiódus alatt. Tekintetbe kell vennünk, hogy ha a mikroutasításban nyílvessző-szimbólum van, az késleltetést jelent a mikroművelet véges átviteli ideje miatt

A regiszterek közötti átvitelben levő késleltetés sokszor előnyös, ha jól használjuk fel. Így például, az alábbi előreszámláló mikroművelet:

$$D \leftarrow \text{countup } D \tag{1.14}$$

amint az 1.2. ábrán látható azt teszi lehetővé, hogy a D regiszter kimenetei áthaladjanak a számláncot vezérlő logikai hálózaton és azután a logikai hálózat kimenetei vezéreljék a D regisztert. Ahhoz, hogy a mikroművelet végrehajtható legyen, akkora késleltetési idő szükséges, ami alatt a jelek végigmennek a regiszter kimenetétől vezető úton a regiszter bemenetéig. Ez a késleltetés lehet a logikai áramkör késleltetése, vagy még gyakrabban az útvonalba beiktatott késleltetőelem miatti késleltetés. Az 1.2*a* és 1.2*b* ábra a D regiszter kimeneti, ill. bemeneti oldalára beiktatott késleltetést mutatja.

A mikroműveleteket a következőképpen lehet osztályozni:

— konstansértékadó mikroműveletek,
— csak átvitelt tartalmazó mikroműveletek,
— unáris mikroműveletek és
— bináris mikroműveletek.

E négy mikroművelettípusra majd később, az 1.3. táblázatban találhatunk példákat. Jegyezzük meg, hogy egy mikroműveletben szereplő konstans oktális vagy bináris szám lehet.

1.2.4. Feltételes mikroutasítások

Ha egy mikroműveletet (vagy mikroműveleteket) csak bizonyos feltétel (feltételek) mellett akarunk végrehajtani, feltételes mikroutasítással írhatjuk azt le. A feltételes mikroutasításoknak két típusa létezik, az IF-THEN (HA-AKKOR) és az IF-THEN-ELSE (HA-AKKOR-EGYÉBKÉNT) típus, melyek a következőkben láthatók:

$$\begin{aligned}&\text{Register G, F(0—5), C(0—14), D(0—14),}\\&\text{IF (G=0) THEN (F} \leftarrow \text{10),}\\&\text{IF (G} \neq \text{1) THEN (C} \leftarrow \text{0, D} \leftarrow \text{0) ELSE (F} \leftarrow \text{11),}\end{aligned} \tag{1.15}$$

A (G=0) vagy (G≠1) feltételeket a feltételes mikroutasításban az = (egyenlő) és a ≠ (nem egyenlő) szimbólumok fejezik ki. Ha a feltétel igaz, a THEN és az ELSE szavak közé (ha van ELSE) zárt mikroutasítások szimultán módon kerülnek végrehajtásra (azaz párhuzamos műveletek). Tehát a mikroutasítások felírási sorrendjének nincs jelentősége. Ha a feltétel nem igaz, az ELSE szót követő mikroutasítás (vagy mikroutasítások) kerül végrehajtásra, és semmi sem történik, ha nincs mikroutasítás az ELSE szót követően.

A feltételes mikroutasításban levő mikroutasítás maga is lehet egy másik mikroutasítás. Például a következő mikroutasításban:

$$\begin{aligned}&\text{Register, A(2—1)}\\&\text{IF (A(1)=0) THEN (A(1)} \leftarrow \text{1)}\\&\qquad\text{ELSE (IF (A(2)=0) THEN (A(2)} \leftarrow \text{1,A(1)} \leftarrow \text{0) ELSE (A} \leftarrow \text{0))}\end{aligned} \tag{1.16}$$

az első ELSE-t követő mikroutasítás egy másik feltételes mikroutasítás.

Az (1.16)-ban szereplő feltételek egyetlen bitre vonatkoznak, azonban a feltételek több bitre is vonatkozhatnak. Például az alábbi mikroutasításban:

```
Register,      S(5—0),    $számláló
IF (S=77₈) THEN (.....)                                    (1.17)
```

az a feltétel, hogy az S számlálót hat biten végzett ellenőrzéssel vizsgálni kell, hogy tartalma 77_8 értékkel egyenlő-e.

A feltétel értéke logikai érték, mivel a feltétel vagy 1 (igaz) vagy 0 (hamis). Ezért logikai operátorokat lehet használni feltételes kifejezések képzésére. Ha például az a feltétel, hogy az X(1) bit legyen 1 és az Y(2) bit legyen 0 és a START kapcsoló legyen az ON állásban, akkor:

```
IF (X(1)=1)*(Y(2)=0)*(START=ON)) THEN (.....)              (1.18)
```

Logikai operátorokkal a feltételes kifejezés még összetettebb is lehet, mint a fenti példában, és hosszadalmassá és nehezen olvashatóvá válhat. Azonban a feltételes kifejezés egyszerűsíthető, ha minden feltételt Boole-változóvá alakítunk át. Például ha a következő helyettesítéseket végezzük el:

```
X(1)=1 vagy X(1)≠0 helyettesítése: X(1),
Y(2)=0 vagy Y(2)≠1 helyettesítése: Y(2)
és
START=ON                helyettesítése: START(ON),
```

ekkor az előbbi mikroutasítás a következő tömör és olvasható utasítás lesz:

```
IF (X(1)*Y(2)*START(ON)) THEN (.....)                      1.19)
```

Ez az egyszerűsítés nem célszerű az olyan esetekben, mint:

```
X(1)≠Y(2),
C=77,                                                      (1.20)
F(0—2)=R(4—6),
```

ahol egyetlen bit kerül egyetlen bittel, vagy egy bittöbbes egy másik bittöbbessel, ill. egy konstanssal összehasonlításra.

1.2.5. Speciális operátorok

Az 1.1. táblázatban található operátorok nem elegendők valamennyi mikroművelet leírására. Ilyen esetben definiálni lehet egy speciális operátort. Mivel valamely operátor egy logikai hálózatot reprezentál, a speciális operátor definíciója egy speciális logikai hálózatot határoz meg. A speciális operátor definiálásához szimbolikus nevet kell választani, amely az összes eddig bevezetett operátor nevétől különböző.

A speciális operátort CDL-utasításokkal definiáljuk. Pl. vegyünk egy kétbites számlálót, amely a következő sorozat szerint számlál: 00, 10, 11, 01, 00,.... Legyen ennek a speciális számlálónak sz2 a neve. Az sz2 operátor definíciója a következő:

```
Operator,      D←sz2 D(0—1),
/begin/        IF (D=00) THEN (D←10),
               IF (D=10) THEN (D←11),                      (1.21)
```

IF (D=11) THEN (D←01),
IF (D=01) THEN (D←00),

end of operator

Az (1.21)-ben a D a *formális változó*, ez képviseli ebben az esetben mind a bemeneteket, mind a kimeneteket. A tulajdonképpeni definíció a **begin** (kezdet) cimke és az **end of operator** utasítás közé van zárva és négy feltételes mikroutasításból áll.

Valamely speciális operátorként használható logikai hálózat definícióját Boole-függvényekkel is ki lehet fejezni. Például vegyünk egy párhuzamos összeadót, amely két, előjel nélküli bináris számot ad össze és (ha van) túlcsordulást jelez. Legyen ennek a speciális operátornak a neve addov. Az addov operátor definíciója ekkor a következő:

Operator,	OV—A(0—23)=X(0—23) addov Y(0—23)	(1.22)
Terminal,	C(23)=0, C(0—22)=X(1—23)∗Y(1—23)+ +Y(1—23)∗C(1—23)+C(1—23)∗X(1—23),	
/begin/	A(0—23)=X(0—23)⊕Y(0—23)⊕C(0—23), OV=X(0)∗Y(0)+Y(0)∗C(0)+C(0)∗X(0),	
	end of operator	

Az (1.22)-ben az X és Y változó a logikai hálózat bemeneteit, az OV és A a kimeneteit képviseli. Terminal- (kapocspár-) utasítást használunk a C(0—23) belső változók definiálására, amelyek a párhuzamos összeadó átvitelei.

1.3. Sorrendbe állítás

Egy mikroművelet egyetlen művelet, viszont egy összetett feladat végrehajtásához több mikroműveletre van szükség. Ahhoz, hogy egy digitális számítógépben nagyszámú mikroművelet sokféle (különböző) funkciót hajtson végre, ezeket a mikroműveleteket vezérelni, szervezni kell, ill. meghatározott sorrendbe kell állítani. A továbbiakban, ezeknek az összetett feladatoknak a megoldására bevezetjük a *cimke*, a *végrehajtási utasítás* és a *vezérlősorrend* fogalmát.

1.3.1. Végrehajtási utasítás

A mikroutasítás nem határozza meg, mikor hajtódjék végre a mikroművelet. A mikroutasítást vezérlőjel kezdeményezi (l. az 1.2. ábrát), amelyet egy *cimke* képvisel. Mivel a vezérlőjel bináris, a cimke logikai változó. Ha a cimke értéke igaz, a mikroutasítások, amelyek a vezérlőjel által vezérelt mikroműveleteket képviselik, végrehajtásra kerülnek. Ha nem igaz akkor semmi sem történik ezekkel.

A *végrehajtási utasítás* cimkéből, és egy vagy több, a cimkéhez tartozó mikroutasításból áll. Felfogható úgy is, mint egy vezérelt mikroutasítás vagy vezérelt mikroutasítások csoportja. Például legyen N az az egybites regiszter, amely parancsjelet ad ki, ha az értéke 1, és legyen P egy egyfázisú órajelgenetátor impulzusa. A vezérlőjel legyen

az N és P közötti logikai ÉS művelet eredménye. Ha az N regiszter kezdetben 1-es értékű volt, a következő utasítások egy vezérelt előreszámláló mikroműveletet illusztrálnak:

```
Register,     D(3—0), N,
Clock,        P,                                    (1.23)
/N*P/         D←countup D.
```

Az (1.23)-ban az utolsó pontban levő utasítás végrehajtási utasítás, amelynek cimkéje ferde zárójelek (/) közé zárt N*P kifejezés. Minthogy az N regiszter kezdeti értéke egy, a D regiszter tartalma csak akkor nő meg eggyel, ha a következő órajel megérkezik, mivel csak ekkor lesz az N*P cimke logikai értéke egy. Minthogy az N regiszter nem változik, a cimke értéke mindannyiszor egy lesz, amikor órajel érkezik, és a D mindannyiszor 1-gyel nő, valahányszor újabb órajel érkezik az áramkörhöz. Azonban, ha az előbbi végrehajtási utasítást a következővel helyettesítjük:

```
/N*P/         D←countup D,    N←0,                  (1.24)
```

a D regiszter tartalma csak egyszer nő meg eggyel, mivel az N regiszter tartalmát 0-ra változtattuk akkor, amikor a D regisztert megnöveltük, és az N*P cimke logikai értéke többé nem lesz egy. Figyeljük meg, hogy az (1.24) utasítás olyan végrehajtási utasítás, amelyben két mikroműveletet egy vezérlőjel vezérel.

1.3.2. Sorrendbe állítás többfázisú órajelgenerátor felhasználásával

Egyszerű digitális számítógépünk utasításlehívó ciklusa három lépést (step) tartalmaz, ez a következőkből látható:

```
step 1,  C←D,
     2,  R←M(C),   D←countup D                      (1.25)
     3,  F←R(OP),  C←R(ADDR)
```

Az első lépés beírja a D regiszter tartalmát a C címregiszterbe. A második lépés a C regiszterben levő címen elhelyezett társzó tartalmát átviszi az R regiszterbe és a D regisztert eggyel megnöveli. A harmadik lépésben az R regiszter műveleti kód (operációs kód) része az F (fázis-) regiszterbe, és az R regiszter címrésze — R(ADDR) — a C regiszterbe kerül át. Szükség van egy *vezérlőjel-sorozat*ra, hogy biztosítsuk a három lépés kívánt sorrendjét. Egy többfázisú órajelgenerátor állítja elő a vezérlősorozathoz szükséges P(1), P(2), P(3) órajeleket. Ha háromfázisú órajelgenerátort használunk e három lépés sorrendbe állításához, az utasításlehívási ciklus a következőképpen alakul:

```
Clock,          P(1—3),
/FETCH*P(1)/    C←D,
/FETCH*P(2)/    R←M(C),     D←countup D,               (1.26)
/FETCH*P(3)/    F←R(OP),    C←R(ADDR),
                End
```

ahol a FETCH szó az utasításlehívási ciklus parancsjele, az end utasítás pedig a sorozat végét jelöli.

1.3.3.
Sorrendbe állítás fázisregiszterrel

Többfázisú órajelgenerátor helyett a vezérlősorozatot egy fázis- (vezérlő-) regiszterrel is elő lehet állítani, amelyhez dekódolót rendelnek. Példaképpen legyen a D egy vezérlőregiszter, a P órajelgenerátor a dekódoló kimeneti kapocspárja: K(0—3). A négy lépésből álló vezérlőjel-sorozatot a következő utasításokkal lehet leírni:

```
Register,      D(0—1),
Decoder,       K(0—3)=D,
Clock,         P,
/K(O)*P/       D←countup D,
/K(1)*P/       D←countup D,
/K(2)*P/       D←countup D,
/K(3)*P/       D←countup D,
               End
```
(1.27)

Az (1.27.) utasításokban leírt vezérlősorozatot órajelgenerátor vezérli, de a vezérlősorozatot a „countup D" mikroművelet lépteti előre. A countup mikroműveletet használtuk, mivel a sorozat minden alkalommal csupán eggyel növeli a D fázisregiszter tartalmát. Ha a vezérlősorozatnak tetszőleges értékek sorozatára van szüksége a D regiszterben, akkor konstansértékadó mikroműveleteket kell használni. Vegyük például az (1.21) utasításaiban leírt számlálósorozatot, amelyet a következő sorozattal lehet megvalósítani:

```
Register,      D(0—1),
Decoder,       K(0—3)=D,
Clock,         P,
/K(0)*P/       D←2,
/K(1)*P/       D←0,
/K(2)*P/       D←3,
/K(3)*P/       D←1,
               End
```
(1.28)

1.3.4.
Sorrendbe állítás többszörösen vezérelt sorozattal

Egy többfázisú vezérlési ciklus olyan utasítások egymásutánja, amelyet valamilyen sajátos vezérlési feladat kivitelezésére alkalmaznak. Minthogy a többfázisú vezérlésre a digitális számítógép valamelyik utasításának végrehajtásához lehet szükség, ezért a digitális számítógép számára egy-egy utasításának végrehajtását többszörösen vezérelt sorozat biztosíthatja. Többszörösen vezérelt sorozat generálásához az egyik lehetőség a fázisregiszterből és többfázisú órajelgenerátorból álló vezérlőáramkör alkalmazása.

Nézzük meg pl. négy, egyenként háromlépéses vezérlősorozat előállítását. E vezérlősorozatok generálását az alábbi utasítások írják le:

```
Register,      F(0—1),
Decoder,       K(0—3)=F
Clock,         P(1—3)
```

```
Comment, itt kezdődik a sorozat, ha F=0.
/K(0)*P(1)/
/K(0)*P(2)/
/K(0)*P(3)/      F←countup F
Comment, itt kezdődik a sorozat, ha F=1.
/K(1)*P(1)/
/K(1)*P(2)/                                          (1.29)
/K(1)*P(3)/      F←countup F
Comment, itt kezdődik a sorozat, ha F=2.
/K(2)*P(1)/
/K(2)*P(2)/
/K(2)*P(3)/      F←countup F
Comment, itt kezdődik a sorozat, ha F=3.
/K(3)*P(1)/
/K(3)*P(2)/
/K(3)*P(3)/      F←countup F
                 End
```

A fenti utasításokban a „comment"-tel kezdődő sort *comment-* (magyarázó-) *utasításnak* nevezik. Éppúgy, mint az előzőekben említett comment kifejezéseket, a comment-utasítást mindehova be lehet iktatni, a leírás érthetőségének (olvashatóságának) javítása érdekében. Az (1.29-ben négy darab, egyenként háromlépéses sorozatból mindegyik egy-egy comment-utasítással kezdődik. A sorozat mindegyikében a három lépést az órajelgenerátor három fázisa lépteti előre. A négy sorozatot a négy *F számlálót növelő* mikroutasítás lépteti előre, mivel azok az F regiszter tartalma szerint a kétbites bináris számok növekvő rendje (00, 01, 10, 11, 00,...) szerint haladnak. Ha ezeket a vezérlősorozatokat tetszőleges sorrendben akarjuk kialakítani, *konstansértékadó* utasításokat kell használni. A többfázisú órajelgenerátor alkalmazása esetünkben ezeknek a sorozatoknak a lépésszámát azonossá teszi. Egyszerű digitális számítógépünkben a többszörös vezérlősorozatok hasonlóak az (1.29)-ben leírtakhoz, azzal az eltéréssel, hogy — amint látni fogjuk — konstansértékadó mikroműveleteket használunk.

1.4.
Tárolt programú számítógép

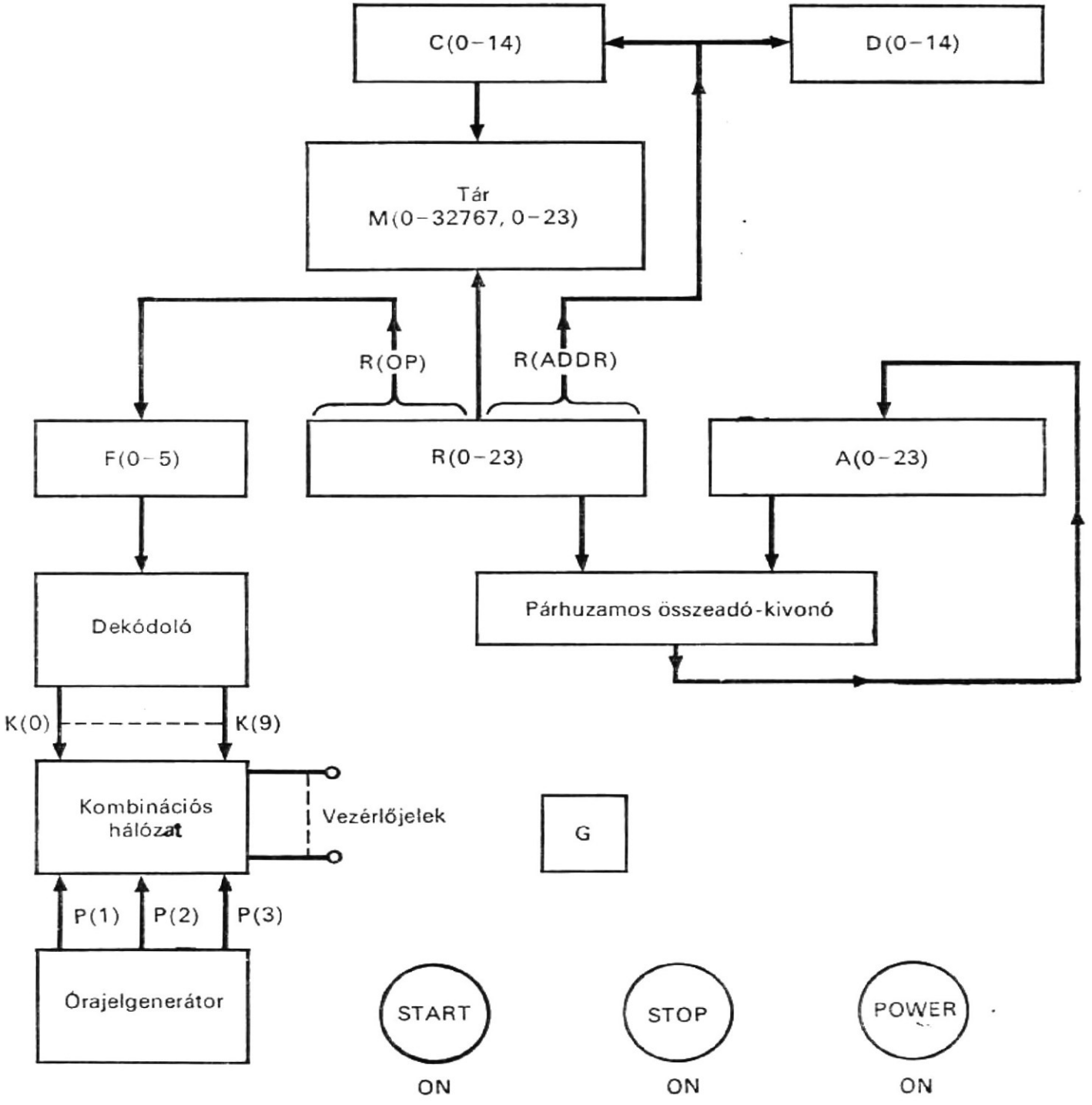
Miután bemutattuk egy egyszerű digitális számítógép funkcionális elemeit, mikroműveleteit és ezek sorrendjeit, most már ismertethetjük a teljes számítógépet, struktúráját, a szám- és utasításábrázolás formátumait, a gépi utasítás készletét, a folyamatábrát és az utasítások végrehajtását. A számítógépnek egy másik változatát is bemutatjuk a speciális operátorok alkalmazásának illusztrálása képpen.

1.4.1. Struktúra

A számítógép logikai kapcsolási rajza az 1.3. ábra hatásvázlatán látható. A struktúrát következő utasításokkal írhatjuk le:

Register,	R(0—23),	$tár tartalom- (adat-) regisztere
	A(0—23),	$akkumulátor
	C(0—14),	$címregiszter
	D(0—14),	$utasításszámláló
	F(0—5)	$fázisregiszter
	G,	$start—stop-vezérlésregisztere
Subregister,	R(OP)=R(0—5),	$az R regiszter op-kód (műveleti kód) része
	R(I)=R(6),	$az R indirekt cimzési bitje
	R(X)=R(7—8),	$az R index bitjei
	R(ADDR)=R(9—23),	$az R címrésze
Memory,	M(C)=M(0—32767, 0—23),	
Decoder,	K(0—9)=F,	
Switch,	POWER(ON)	$a számítógép működésének kezdeményezése (tápfeszültség bekapcsolása)
	START(ON),	$a számítógép működésének indítása
	STOP(ON),	$a számítógép működésének leállítása
Terminal,	ADD=K(0),	$összeadás- (add) parancs (1.30)
	SUB=K(1),	$kivonás- (subtract) parancs
	JOM=K(2),	$ugrás mínuszra (jump-on-minus) parancs
	STO=K(3),	$tárolás (store-accumulator) akkumulátorban parancs
	JMP=K(4),	$feltétel nélküli ugrás (jump) parancs
	SHR=K(5),	$léptetés jobbra egy bittel parancs
	CIL=K(6),	$ciklikus léptetés balra egy bittel parancs
	CLA=K(7),	$akkumulátor- (clear accumulator) törlés parancs
	STP=K(8),	$stop parancs
	FETCH=K(9),	$utasításlehívási ciklus parancs (fetch)
Clock,	P(1—3),	$háromfázisú óra

A bemutatott konfiguráció tárolóegységből, vezérlőegységből és aritmetikai egységből áll. A tárolóegység az M tárat (memóriát) az R tartalom- (adat-) regisztert és a C címregisztert tartalmazza. A vezérlőegység az F fázisregiszterből és a hozzá tartozó dekódolóból, a D utasításszámlálóból, a G start—stop-vezérlésregiszterből, a P órából és a POWER, START és STOP kapcsolókból tevődik össze. A dekódoló K(0—9) kimenetei adják a parancsjeleket. Ha a G regiszter értéke 1, a számítógép „működési" állapotban; egyébként pedig „várakozás" állapotban van. Bekapcsolásakor a POWER kapcsoló kezdeményezi a számítógép működését, a START kapcsoló megindítja és a STOP kapcsoló leállítja a gépet. Az aritmetikai egység az A regiszterből, az R regiszterből és egy 24-bites párhuzamos összeadó—kivonó-egységből áll, amely hozzáadja az R regiszterben levő számot az A regiszterben levőhöz, ill. kivonja az R regiszterben levőt az A regiszter tartalmából. Ez az összeadó- és kivonóegység úgy kezeli



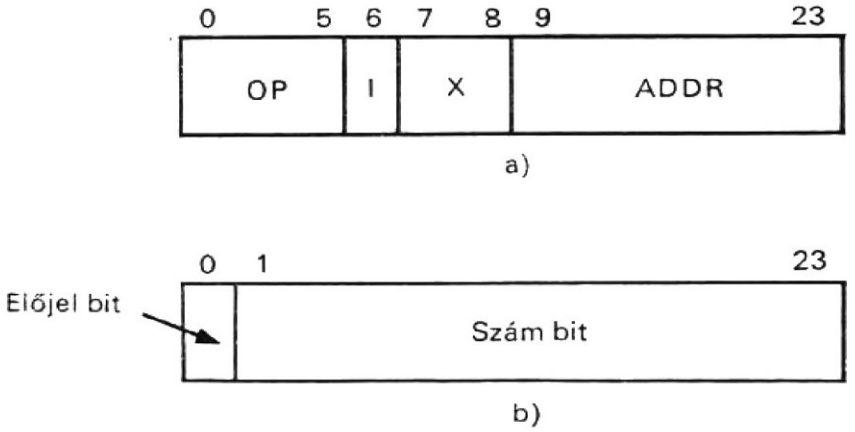
1.3. ábra
Egyszerű, tárolt programú számítógép tömbvázlata

a számokat, mintha azok előjel nélküli bináris számok lennének, de a túlcsordulást nem jelzi, és nem is kezeli.

Az (1.30) utasításban az R regiszter négy részregiszterét deklaráltuk; ezek mindegyike a gépi utasítás egy-egy mezejét reprezentálja. Minthogy az indexelt és az indirekt címzést később tárgyaljuk, ezzel a két mezővel most nem foglalkozunk. Az (1.30) utasításban a dekódoló valamennyi kimenetének adtunk egy szimbolikus nevet, melyek a parancshoz tartozó műveletet vezérlik.

1.4.2.
Utasítás- és adatábrázolás

A számítógép valamennyi szava 24 bitből áll. Ha a szó utasítás, az első hat bit a műveleti kód (op-kód) mezőt, egy bit az indirekt címzési mezőt, két bit az indexezési mezőt és 15 bit a címmezőt tartalmazza. Ha a szóban szám helyezkedik el, 23 bit a számbitet és egy bit az előjelet reprezentálja. A számbitek tört bináris számot reprezentálnak, ahol a bináris pont az előjelbit és a legértékesebb számbit között van. A számok ábrázolása előjeles kettes komplemens [1]. Az utasítás- és adatformátumok az 1.4. ábrán láthatók.



1.4. ábra
Szóformátumok
a) **gépiutasítás-formátum;** *b)* **számformátum**

1.4.3.
A gépiutasítás-készlet

Kilenc gépi utasítás van: ADD, SUB, FOM, STO, JMP, SHR, CIL, CLA és STP. Ezek az utasítások és műveleti kódjaik az 1.2. táblázatban láthatók, ahol m az utasítás címmezejében található tárcímet reprezentálja. Ez a tárcím utasításnak is, operandusnak is címe lehet. Az SHR utasítás az akkumulátorban levő szót egy bittel lépteti jobbra, miközben egy nullát iktat be a bal oldali legelső bit helyére. Az CIL utasítás egy bittel balra körbelépteti az akkumulátorban levő szót. Az STP utasítás leállítja a számítógép működését. Ez a három utasítás nem igényli az *m* tárcimet.

1.2. táblázat
A gépiutasítás-készlet

Gépi utasítás neve	Szimbolikus kód	Műveleti kód (op-kód)
Összeadás (Addition)	ADD m	00
Kivonás (Subtraction)	SUB m	01
Ugrás mínuszra (Jump on minus)	JOM m	02
Tárolás (Store)	STO m	03
Ugrás (Jump)	JMP m	04
Léptetés jobbra (Shift right)	SHR	05
Ciklikus léptetés balra (Circular left shift)	CIL	06
Törlés és összeadás (Clear add)	CLA m	07
Stop	STP	10

Megjegyzés: az *m* tárcímet jelöl. A műveleti kód oktális.

Az ADD és SUB utasítás összeadja, ill. kivonja az m címen levő tároló helyen található számot az akkumulátor tartalmával (ból). A CLA utasítás ugyanúgy működik, mint az ADD, kivéve, hogy az akkumulátor törlődik, mielőtt az összeadás végbemenne. A STO utasítás az akkumulátorban levő számot az m című tároló helyre írja be. A felsoroltak mind operandus címet igényelnek.

A JMP utasítás a következő utasítást veszi az m című tároló helyről; ezt teszi a JOM utasítás is, ha az akkumulátorban levő szám negatív. Ezekhez a gépiutasításokhoz tartozó cím: m.

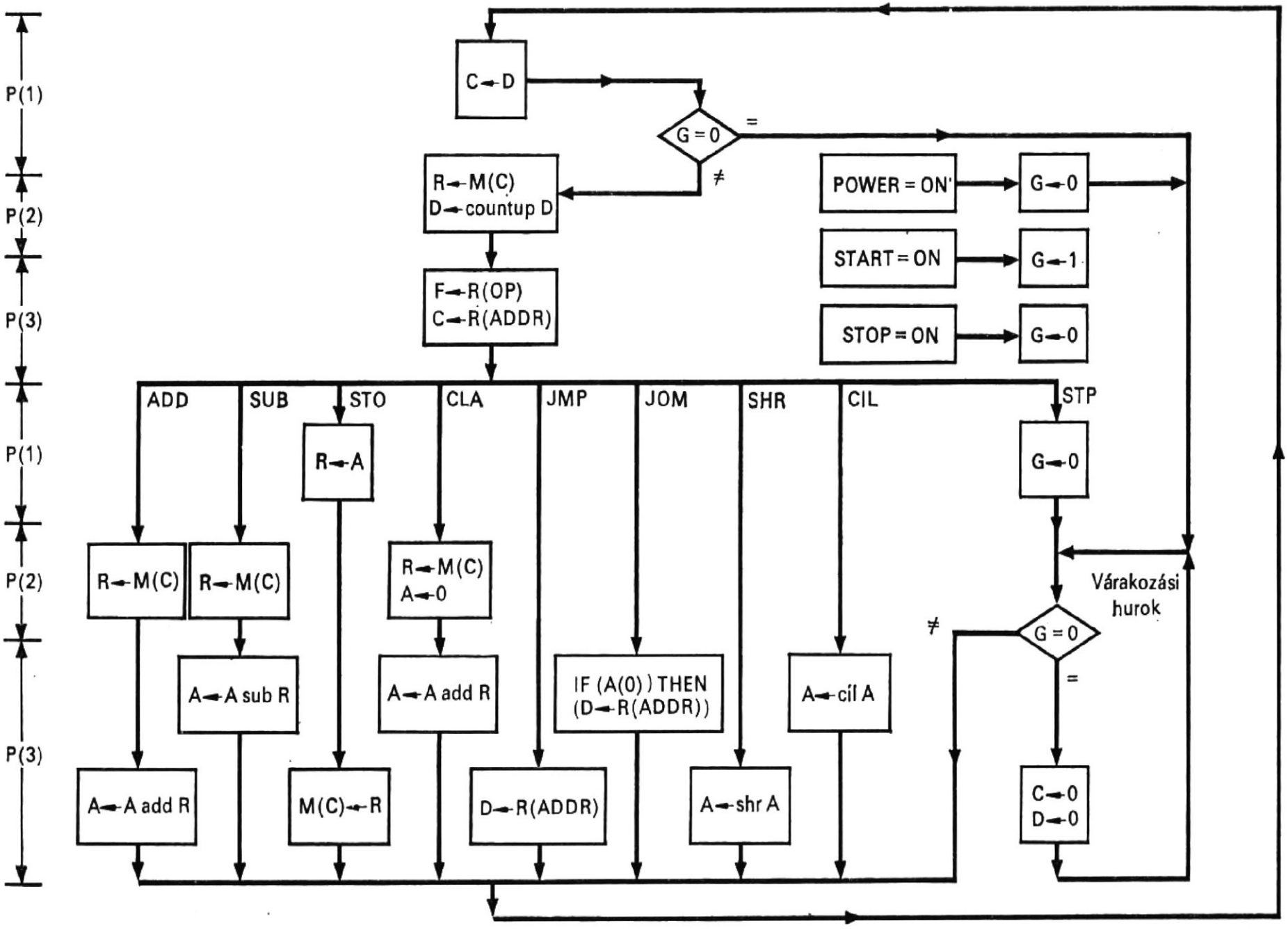
1.4.4. Folyamatábra

A számítógép az egyik vezérlési ciklust a másik után hajtja végre. Az egyes vezérlési ciklusok alatt egy, a memóriában tárolt gépi utasítás kerül végrehajtásra. Az utasítások tehát egymás után hajtódnak végre, a tárcímek növekvő sorrendjében. Az utasítás végrehajtásnak ezt a sorrendjét *normál sorrendnek* nevezik.

A vezérlési ciklus alatt a következők mennek végbe: az utasításlehívási sorozat, amelyet az utasításvégrehajtási sorozat követ. Azért, hogy az utasítás lehívási ciklusában a gép kiadhassa a végrehajtandó gépi utasítást, dekódolja a műveleti kódot, átviszi az operandus címét a címregiszterbe és létrehozza a következő gépi utasítás címét. Ezeket a mikroműveleteket már leírtuk az (1.26) utasításokkal. Az utasításvégrehajtási ciklusban a gépi utasítás által specifikusan igényelt mikroműveleteket hajtja végre. Utasításlehívási ciklus csak egy van, mivel az egyes gépi utasítások lehívásához szükséges mikroműveletek ebben a fázisban egyformák; azonban kilenc utasításvégrehajtási sorozat van, mind a kilenc gépi utasításhoz egy-egy. Röviden, a számítógép a következőképpen végzi a végrehajtást: kezdi az utasításlehívási sorozattal, folytatja az utasításvégrehajtási sorozattal, majd váltakozva ismétli ezeket.

A számítógép szekvenciális működését az 1.5. folyamatábra mutatja. Az ábra felső része az utasításlehívási ciklushoz szükséges egyetlen utat mutatja, míg az alsó rész kilenc párhuzamos ágból áll, a kilenc utasítás végrehajtási ciklusa számára. Ezek az ágak konvergálnak és az utasításlehívási ághoz térnek vissza. Amikor a POWER kapcsolót az ON pozícióba kapcsolják, a G regiszter 0 értéket kap és a számítógép a *várakozó állapotban* van, mialatt a C és D regiszterek folyamatosan a 0 értéken maradnak. Figyeljük meg a várakozás hurkot az ábra jobb oldali alsó sarkában. Amikor a START gombot megnyomják, a G regiszter 1 értéket kap, a számítógép belép a *működési állapotba* és megkezdi az utasításlehívási ciklus végrehajtását. Az első végrehajtandó gépi utasítás a 0 címen van, mivel a címregiszter 0-ra lett állítva a várakozó állapot alatt. Amint a folyamatábrából megfigyelhetjük, a számítógép mindaddig folytatja vagy az utasításlehívási ciklus, vagy valamelyik utasításvégrehajtási ciklus végrehajtását, amíg STP utasítással nem találkozik, vagy a STOP gombot meg nem nyomják; ekkor, a két eset közül bármelyikben a számítógép belép a várakozó hurokba. Az utasításlehívási ciklusban ellenőrzésre kerül a G regiszter tartalma. Ha a G értéke 1, folytatódik az utasításvégrehajtás; egyébként a végrehajtás belép a várakozás hurokba.

Az utasításlehívási ciklus végén a gépi utasítás műveleti kódja az F (fázis-) regiszterben van. A dekódoló segítségével aktivizálódik a megfelelő parancsjel és a speciális utasításvégrehajtási ciklus megkezdődik. Amint az a folyamatábrán látható, az egyes végrehajtási sorrendek egy vagy két lépésből állnak. Feltételezzük, hogy a főtárciklus kiolvasási része a P(2) órajel alatt, míg a beírási része a P(3) órajel folyamán történik. A kilenc végrehajtási sorozat a következő: ADD, SUB, STO, CLA, JMP, JOM, SHR, CIL és STP. Az ADD sorozat két lépésből áll: a szó átviteléből a tárból az R regiszterbe, majd az A regiszter tartalmához történő hozzáadásából. Mind a SUB, mind a CLA soro-



1.5. ábra
Az egyszerű, tárolt programú számítógép folyamatábrája

zat hasonló az ADD sorozathoz, azzal a kivétellel, hogy a SUB sorozatban kivonás történik és a CLA sorozatban az A regisztert előbb 0 értékre állítják. A STO sorozat az A regiszter tartalmát átviszi az R regiszterbe, azután a főtárban tárolja azt. Az SHR és az CIL egy bittel jobbra léptetési, ill. egy bittel ciklikus balra léptetési mikroműveleteket hajt végre. A JMP sorozat az R regiszterben levő címrészt átviszi a D regiszterbe úgy, hogy ez a cím lesz a következő utasítás címe. A JOM sorozat ugyanazt végzi, mint a JMP sorozat, ha az A regiszterben levő szám negatív. Tehát az JMP sorozat megszakítja az utasításvégrehajtás normál sorrendjét; ugyanezt teszi a JOM is, feltétel beteljesülése esetén.

1.4.5. Utasításokkal végzett leírás

Többszörös vezérlési ciklus segítségével a folyamatábra könnyen átírható utasításos leírásba. A többszörös vezérlési elv azonos az (1.29) utasításban leírttal, kivéve, hogy az F (fázis-) regiszter vezérlésére konstansértéket adó mikroműveletet használunk. Az egyszerű, tárolt programú számítógép leírása utasításokkal:

```
Comment, indexek és a konstansok decimálisak.

Comment, következnek a start—stop mikroműveletek.
/POWER(ON)/        G←0, F←8          $F beállítása indítja a stop-sorozatot
/START(ON)*P(2)/G←1,
/STOP(ON)/         G←0,

Comment, következik az utasításlehívási (FETCH) ciklus.
/FETCH(1)/         C←D, IF (G=0) THEN (F←8),
/FETCH(2)/         R←M(C), D←countup D,
/FETCH(3)/         F←R(OP), C←R(ADDR),

Comment, kezdődik az összeadás végrehajtása.
/ADD*P(2)/         R←M(C),
/ADD*P(3)/         A←A add R, F←9,

Comment, következik a kivonás végrehajtása.
/SUB*P(2)/         R←M(C),
/SUB*P(3)/         A←A sub R, F←9,

Comment, következik a tárolás (store) végrehajtása.
/STO*P(1)/         R←A,
/STO*P(3)/         M(C)←R, F←9,

Comment, következik a törlés és összeadás (clear add) sorozat.
/CLA*P(2)/         R←M(C), A←0,
/CLA*P(3)/         A←A add R, F←9,

Comment, kezdődik a stop-sorozat.                                    (1.31)
/STP*P(1)/         G←0,
/STP*P(3)/         IF (G=0) THEN (C←0, D←0) ELSE (F←9),

Comment, a többi sorozat.
/JMP*P(3)/         D←R(ADDR), F←9,
/JOM*P(3)/         IF (A(0)) THEN (D←R(ADDR)), F←9,
/SHR*P(3)/         A←shr A, F←9,
/CIL*P(3)/         A←cil A, F←9,
                   End
```

1.3. táblázat

A számítógép mikroműveletei, ellenőrző feltételei, vezérlőjelei és kézi vezérlőjelei

1. *Konstans-értéket adó mikroműveletek*

G←0
G←1
C←0
D←0
A←0
F←8
F←9

2. *Unáris mikroműveletek*

D←countup D
A←shr A
A←cil A

3. *Csak átvitelt tartalmazó mikroműveletek*

C←D
D←R(CIM)
D←R(CIM)
F←R(OP)
R←A
R←M(C)
M(C)←R

4. *Bináris mikroműveletek*

A←A add R
A←A sub R

5. *Ellenőrző feltételek*

G=0
G=1
A(0)=1

6. *Vezérlőjelek*

K(0)∗P(2)

K(9)∗P(3)

7. *Kézi vezérlőműveletek*

POWER←ON
START←ON
STOP←ON

1.4.6.
Adat- és vezérlési utak

Az adat- és vezérlési utak huzalok, amelyek számítógépelemeket kötnek össze. Ezek azok a huzalok, amelyek a digitális számítógép hátlapjain találhatók. Egyetlen huzal szakadása a számítógép hibás működését okozhatja. Ezek a huzalok a hardware-nek azt a részét képezik, amely a mikroműveleteket, az ellenőrző *(test)* feltételeket, a vezérlőjeleket és a kézi vezérlési műveleteket valósítja meg (l. az 1.3. táblázatot).

Az 1.3. táblázat hét csoportból áll. Az első csoport az értékadó mikroműveleteket tartalmazza, mint amilyen például a G regiszter törlése vagy az F regiszter oktális 10-re állítása.

A második csoport a léptető és növelő (inkrementáló) mikroműveletekből áll, amelyek unáris operátorokat igényelnek. Az első csoportban hét mikroművelet van, a másodikban három. Az utóbbi mikroműveletek mindegyike csak egy regisztert tételez fel; az elemek között nincs szükség vezérlésre.

A harmadik csoport csak átvitelt tartalmazó mikroműveletekből áll, amilyenek például a regiszter és a tár tartalom átvitele.

A negyedik csoport aritmetikus mikroműveleteket tartalmaz, ilyen esetünkben az összeadás és a kivonás. Ezek a mikroműveletek az add és sub bináris operátorokat igénylik. Minthogy ezek a mikroműveletek két vagy több regisztert tételeznek fel, az elemek között szükséges a huzalozás. Ha ezek az átvitelek párhuzamosak, mint az 1.3. táblázatban szereplő műveletek, a huzalok száma arányos az átvitelre kerülő párhuzamos bitek számával.

Az ötödik csoportot az ellenőrző feltételek alkotják. Például ellenőrizni lehet, hogy G egyenlő-e 0-val, vagy az akkumulátorban levő szám negatív-e.

A hatodik csoport vezérlőjelekből áll, míg a hetedik csoport kézi vezérlési műveleteket tartalmaz. Az ötödik csoportban három mikroművelet van, 17 vezérlőjel a hatodik csoportban és három kézi vezérlési művelet a hetedik csoportban. Minthogy az ellenőrző feltétel, a vezérlőjel és a kézi vezérlési műveletek binárisak, csak egy-egy huzal szükséges mindegyikhez. Ezek a huzalok sok esetben elég hosszúak, mivel az ellenőrzési feltétel távol helyezkedhet el attól a helytől, ahol a mikroművelet végbemegy.

A számítógép adatútjait az 1.3. ábrán látható összekötések mutatják, amelyek a regiszterek és a főtár közötti kapcsolatot illusztrálják. A vezérlési utakat — az áttekinthetőség érdekében — nem tüntettük fel. Ezen utak kiválasztása korlátozza a megengedett mikroműveletek típusát. Nagyon fontos, hogy ezen utak számát a lehető legkisebbre válasszuk még azon az áron is, hogy az egyes ciklusokban esetleg így több lépésre van szükség.

1.4.7.
Példa speciális operátorok használatára

Az add és sub bázisoperátorok előjel nélküli bináris számokat adnak össze, ill. vonnak ki. Előjeles, 2-komplemens ábrázolású bináris számok [1] összeadásakor és kivonásakor az előjelbitet számbitként kezelik. Ezért ezt a két bázisoperátort fel lehet használni előjeles, 2 komplemens kódú számok bináris összeadására és kivonására is. Viszont az add és sub bázisoperátor leírása elhanyagolja a túlcsordulást. Ha a túlcsordulást fel kell tüntetni, speciális operátort kell használni az összeadásra és a kivonásra. Például,

leírjuk az addov és subov speciális operátorokat[5]. Ezek az operátorok helyettesítik az add és sub bázisoperátorokat az (1.31) utasításokban. Eszerint az a tárolt programú számítógép, amely az addov és subov speciális operátorokat alkalmazza, a következő utasításokkal írható le:

```
Register,      R(0—23)          $tár tartalom regisztere
               A(0—23),         $akkumulátor
               C(0—14),         $címregiszter,
               D(0—14),         $utasításszámláló
               F(0—5),          $fázisregiszter
               G,               $start—stop vezérlési
               OV,              regiszter
                                $túlcsordulás- (overflow)
                                indikátor
Subregister,   R(OP)=R(0—5),
               R(ADDR)=R(9—23),
Memory,        M(C)=M(0—32767, 0—23),
Decoder,       K(0—19)=F,
Switch,        POWER(ON)        $kezdeményezi a számító-
                                gép működését
               START(ON),       $indítja a számítógép mű-
                                ködését
               STOP(ON),        $leállítja a számítógép mű-
                                ködését
Light,         LTOV(ON, OFF)    $túlcsordulásindikátor
                                (overflow light)
Terminal,      ADD=K(0),        $összeadássorozat parancs
               SUB=K(1),        $kivonássorozat parancs
               JOM=K(2),        $ugrás mínuszra parancs
               STO=K(3),        $akkumulátor tárolás
                                parancs                 (1.32)
               JMP=K(4),        $ugrásparancs
               SHR=K(5),        $léptetés jobbra egy bittel
                                parancs
               CIL=K(6),        $ciklikus léptetés balra egy
                                bittel parancs
               CLA=K(7),        $akkumulátortörlés parancs
               STP=K(8),        $stop-parancs
               FETCH=K(9),      $utasításlehívás ciklus
Clock,         P(1—3),
```

Comment, itt kezdődnek a start—stop mikroműveletek.

```
/POWER(ON)/    G←0, F←8, OV←0, LTOV←OFF,
/START(ON)/    G←1,
/STOP(ON)/     G←0,
```

Comment, itt kezdődik az utasításlehívási sorozat.

```
/FETCH*P(1)/   IF (G=0) THEN (F←8),
               IF (ON=1) THEN (LTOV←ON, G←0, F←8)
               ELSE (C←D),
```

Az addov és subov speciális operátorok már nem részei a nyelvnek, hanem definiált egységek, ezért ezek esetében már nem feltétlenül szükséges ragaszkodnunk az eredeti elnevezéshez. ***(Fordító)***

```
/FETCH * P(2)/      R ← M(C), D ← countup D,        $tár szó rendelkezésre áll
                                                    P(2)-nél
/FETCH * P(3)/      F ← R(OP), C ← R(ADDR),
```

Comment, itt kezdődik az összeadás végrehajtása.

```
/ADD * P(2)/        R ← M(C),
/ADD * P(3)/        OV — A ← A addov R, F ← 9,
```

Comment, itt kezdődik a kivonás végrehajtása.

```
/SUB * P(2)/        R ← M(C),
/SUB * P(3)/        OV — A ← A subov R, F ← 9,
```

Comment, itt kezdődik a tárolás végrehajtása.

```
/STO * P(1)/        R ← A,
/STO * P(3)/        M(C) ← R, F ← 9,            $akkumulátor tartalma tá-
                                                rolva van P(3)-nál
```

Comment, itt kezdődik a törlés- és összeadássorozat.

```
/CLA * P(2)/        R ← M(C), A ← 0,
/CLA * P(3)/        OV — A ← A addov R, F ← 9,
```

Comment, itt kezdődik a stop-sorozat.

```
/STOP * P(1)/       G ← 0,
/STOP * P(3)/       IF (G = 0) THEN (C ← 0, D ← 0) ELSE (F ← 9)
```

Comment, itt következik a többi sorozat.

```
/JMP * P(3)/        D ← R(ADDR), F ← 9,
/JOM * P(3)/        IF(A(0)) THEN (D ← R(ADDR)), F ← 9,
/SHR * P(3)/        A ← shr A, F ← 9,
/CIL * P(3)/        A ← cil A, F ← 9,
                    End
Operator,           R(OV, 0 — 23) = A(0 — 23) addov B(0 — 23),
```

Comment, mind az A, mind a B regiszterben levő számok ábrázolása előjeles 2-komplemens. A B regiszter tartalmát az A regiszteréhez adjuk hozzá. Az OV 1 értéket kap túlcsordulás esetén.

```
Terminal,           C(23) = 0,
                    C(0 — 22) = A(1 — 23) * B(1 — 23) + B(1 — 23) * C(1 — 23) +
                      + C(1 — 23) * A(1 — 23),
                    SUM(0 — 23) = A(0 — 23) ⊕ B(0 — 23) ⊕ C(0 — 23),
/begin/             R(0 — 23) = SUM(0 — 23),
                    R(OV) = A(0) * B(0) * C(0)' + A(0)' * B(0)' * C(0),
                    End of operator
Operator,           R(OV, 0 — 23) = A(0 — 23) subov B(0 — 23),
```

Comment, mind az A, mind a B regiszterben levő számok ábrázolása előjeles 2-komplemens. A B regiszter tartalmát vonjuk ki az A regiszteréből. Az OV 1 értéket kap túlcsordulás esetén,

```
Terminal,           C(23) = 1,
                    C(0 — 22) = A(1 — 23) * B(1 — 23)' + B(1 — 23)' * C(1 — 23) +
                      + C(1 — 23) * A(1 — 23),
```

```
                    SUM(0—23)=A(0—23)+B(0—23)'+C(0—23),
/begin/             R(0—23)=SUM(0—23),
                    R(OV)=A(0)*B(0)'*C(0)'+A(0)'*B(0)*C(0),
                    End of operator
```

Az (1.32) utasításaiba pótlólag beiktattuk az OV regisztert és a LTOV kijelzőt, a túlcsordulás indikálására. Ezt a regisztert és a kijelzőt ugyancsak a POWER kapcsoló helyezi üzemkész állapotba. Továbbá az OV regiszter állapota az utasításlehívási ciklus első lépése során kerül ellenőrzésre, miközben a G regiszter ellenőrzése folyik. Ha az OV regiszter tartalma 1, a számítógépnek várakozási állapotba kell lépnie. Az addov operátor cseréli fel az add mikroutasítást az ADD és CLA sorozatokban, míg a subov operátor a sub mikroutasítást helyettesíti a SUB sorozatban. Az összes többi utasítás változatlan marad.

A két speciális operátor definíciója a számítógép leírása után következik. A bemenetek 24-bitesek, ezeket az A és B reprezentálja, míg a kimenetek 25-bitesek, ezt R reprezentálja. Az A, B és R formális változók. A SUM(0—23) és C(0—23), sorrendben a kapuzott átvitelű párhuzamos összeadó [1] összegbitjeit és átvitelbitjeit reprezentálják az addov operátor definíciójában. Ugyancsak ezek reprezentálják a subov operátor definíciójában a különbségbiteket és átvételbiteket. Összeadás közben akkor keletkezik túlcsordulás, ha mindkét előjel pozitív és a C(0) 1-gyel egyenlő, vagy ha mindkét előjel negatív és a C(0) értéke 0. Kivonáskor túlcsordulás következik be, ha a két előjel különböző. Túlcsordulás lesz, ha C(0) és B(0) egyaránt 1 vagy 0.

Feladatok

1.1. Adott az SR(0—35) regiszter, deklaráljunk részregiszter-utasítással egy SR(0) részregisztert az előjelbit számára, egy SR (1—8) részregisztert a karakterisztikát tartalmazó biteknek és egy SR(9—35) részregisztert a mantisszabiteknek.

1.2. Adott egy dekódoló, amelyet a következő utasítások írnak le:

```
Register,   C(1—3),
Decoder,    N(0—7)=C,
```

használjunk terminal- (kapocspár-) utasítást a dekódolónak a *decoder* utasítás helyetti leírására!

1.3. Adott az N(EGY, KETTŐ, HÁROM, OFF) négyállású kapcsoló és három indikátor: INDEGY(ON, OFF), INDKETTŐ(ON, OFF) és INDHÁROM(ON, OFF). Ha az N kapcsoló OFF állásban van, egyik indikátor sincs ON állapotban. Amikor az N kapcsoló az EGY állásban van, csak az INDEGY indikátor van ON állapotban. Amikor a kapcsoló a HÁROM állásban van, mind a három indikátor ON állapotba kerül. Írjuk le az ismertetett kapcsoló (indikátor) működésének logikai vázlatát mikroutasításokkal!

1.4. Az egybites teljes összeadó—kivonó olyan logikai hálózat, amely egyetlen bit összeadását vagy kivonását végzi, ha — megfelelően — a hozzárendelt egybites AS regiszter 0 vagy 1 állapotban. Három bemenete van:

— összeadandó vagy kisebbítendő,
— összeadandó vagy kivonandó és
— bemeneti átvitel vagy átvétel.

Két kimenete van: összeg vagy különbség, és kimeneti átvitel vagy kimeneti átvétel. Írjuk le terminal- (kapocspár-) utasítással a teljes összeadó—kivonót!

1.5. Az A és B egybites regiszterek tartalmát aritmetikailag kell összeadni és az eredménybitet és az átvitelbitet a C, ill. D egybites regiszterekben kell tárolni. Használjunk feltételesen mikroutasításokat az összeadás leírására!

1.6. Adjunk numerikus példát a túlcsordulás két lehetséges esetére, ha két — előjeles — 2-komplemenses ábrázolású bináris szám kerül összeadásra!

1.7. Ismételjük meg az 1.6. feladatot, ha egy bináris számot kivonunk egy másikból!

1.8. Helyettesítsük az (1.29) utasításokban levő növelési (increment) mikroműveleteket konstansértékadó mikroműveletekkel, ha az (1.29) utasításaiban leírt négy sorozatot az F regiszter tartalma szerint léptetjük előre, a következő sorrendben: 00, 10, 01, 11, 00,....

1.9. Ha a LTSTOP(ON, OFF) kijelzőt pótlólagosan berakjuk az (1.30) utasításokban leírt számítógépbe, módosítsuk az (1.30) és (1.31) utasításokat úgy, hogy az indikátor ON vagy OFF állapotba kerül, ha a G regiszter tartalma 0, ill. 1.

1.10. Ha a K(10) dekódoló kimenetét úgy választottuk ki, hogy azt a számlálósorozatot vezérelje, amely az akkumulátorban levő 1-esek számát számlálja. Írjuk le a számlálósorozatot és helyezzük ezt el az 1.5. ábra folyamatábrájába, és iktassuk be az (1.31) utasításleírásba! Ha szükséges pótlólagos elemek is alkalmazhatók.

1.11. Szimuláljuk az (1.30) és (1.31) utasításokkal leírt, tárolt programú számítógépet algoritmikus nyelvvel, pl. FORTRAN, ALGOL vagy PL/1 nyelvvel!

Irodalom

[1] CHU, Y.: *Digital Computer Design Fundamentals.* New York, McGraw-Hill Book Company, 1962.

[2] IVERSON, K. E.: *A Programming Language.* New York, John Wiley & Sons, Inc., 1962.

[3] SCHORR, H.: *A Register Transfer Language to Describe Digital Systems.* Technical Report No. 30, Digital Systems Laboratory, Department of Electrical Engineering, Princeton University, September, 1962.

[4] FALKOFF, A. D.—IVERSON, K. E.: *A Formal Description of System/360.* IBM Systems Journal, 3, No. 3, 1964.

[5] SCHLAEPPI, H. P.: *A Formal Language for Describing Machine Logic, Timing and Sequencing (LOTIS.)* IEEE Trans. on Electronic Computers, August, 1964, pp. 439...448.

[6] MULLERY, A. P.: *A Procedure Oriented Machine Language.* IEEE Trans. on Electronic Computers, August, 1964, pp. 449...455.

[7] CHU, Y.: *An Algol-like Computer Design Language.* Comm. of ACM, October, 1965, pp. 607...615.

[8] PARNAS, D. L.: *A Language for Describing the Function of Synchronous Systems.* Comm. of ACM, February, 1966, pp. 72...76.

[9] WILBER, J. A.: *A Language for Describing Digital Computer.* Report No. 197, Department of Computer Science, University of Illinois, February 15, 1966.

[10] GIESE, A.: *Hargol-A Hardware Oriented Algol Language.* Internal Report No. VA5, August, 1966, A/S Regnecentralen, Copenhagen, Denmark.

[11] METZE, G.—SESHU, S.: *A Proposal for a Computer Compiler.* Proc. of the SECC Conference, 1966, pp. 253...263.

[12] ZUCKER, M. S.: *LOCS: An EDP Machines Logic and Control Simulator.* International Convention Record, Part 3, 1965, pp. 28...50.

[13] CLARK, W. A.: *Macromodular Computer Systems.* AFIPS Spring Joint Computer Conference, 30, pp. 335...336, 1967.

[14] ORNSTEIN, S. M.—STUCKI, M. J.—CLARK, W. A.: *A Functional Description of Macromodules.* AFIPS Spring Joint Computer Conference, 30, pp. 337...364, 1967.

[15] FRIEDMAN, T. D.: *ALERT: A Program to Compile Logic Designs of New Computers.* Digest of the First Annual IEEE Computer Conference, September 6, 1967, pp. 128...130.

[16] GORMAN, D. E.: *Systems Level Design Automation: A Progress Report of the System Descriptive Language (SDL11)*. Digest of the First Annual IEEE Computer Conference, September 6, 1967, pp. 131...134.

[17] DARRINGER, J. A.: *A Language for the Description of Digital Computer Processor*. Proceedings for the Design Automation Workshop, July, 1968.

[18] DULEX, J. R.—DIETMEYER, D. L.: *A Digital Systems Design Language (DDL)*. IEEE Transactions on Computers, September, 1968, pp. 850...861.

[19] CHU, Y.: *Introduction to Computer Organization*. Englewood Cliffs, N. J., Prentice-Hall, Inc., 1970.

2.
CDL alkalmazása vezérlési feladatok szimbolikus leírására

Az 1. fejezetben egy egyszerű, tárolt programú számítógép leírásával bemutattuk a CDL (számítógéptervezési) nyelvet. E fejezetben, a CDL alkalmazására további példaként néhány egyszerű feladat megoldásával ismertetjük meg az Olvasót. A példákon keresztül világossá válik majd, hogy mind a hardware, mind a software algoritmusokat valósít meg, azonban a korlátaik különbözők. Egy számítógéprendszerben a hardware és a software közötti egyensúly állandóan változik az áramköri technológia és az alkalmazások előrehaladásával.

2.1.
Soros paritásgenerátor

A bináris szóban a paritás- (párosság-) bit egy külön, további bit, amelynek feladata egyetlen olyan hiba ellenőrzése, amely áramkör meghibásodása következtében jön létre. Ez a hibaellenőrzés lehet páratlan vagy páros. Vizsgáljuk meg a bináris szó paritásbitjének soros üzemmódú generálását!

2.1.1.
Soros paritásbit generálása

Tételezzünk fel egy bináris szót az A(1—6) jelű 6-bites regiszterben, amelynek paritásbitjét a PB, egybites regiszter tárolja. Ha a paritásbitet úgy választjuk meg, hogy az A-PB regiszterben láncban (sorban) az 1-esek száma páratlan, akkor a párosságbitet *páratlan paritásbitnek* nevezik. Fordítva, ha a paritásbit megválasztása olyan, hogy az 1-esek száma páros, akkor a párosságbitet *páros paritásbitnek* nevezik. Pl., ha az A-PB regisztersor tartalma 0010101, a regisztersorban az 1-esek száma három, vagyis páratlan. Páratlan paritásbit előállításához a PB regiszter helyesen tartalmaz 1-est. Páros paritás esetén a PB-nek 0-t kell tartalmaznia ahhoz, hogy a regiszterben sorban az 1-esek száma páros, vagyis kettő legyen.

Az A-PB regiszterben láncban levő bináris szó páratlan paritásbitjét az előző definíció szerint a következő egyenlet határozza meg:

$$A(1)\oplus A(2)\oplus A(3)\oplus A(4)\oplus A(5)\oplus A(6)\oplus PB=1, \tag{2.1}$$

ahol a $\oplus$ szimbólum a logikai KIZÁRÓ—VAGY operátort jelöli. A következő két Boole-reláció felhasználásával:

$$PB\oplus PB=0 \tag{2.2}$$

és

$$1\oplus 1=0 \tag{2.3}$$

a (2.1) egyenlet átalakításával:

$$PB=A(1)\oplus A(2)\oplus A(3)\oplus A(4)\oplus A(5)\oplus A(6)\oplus 1 \tag{2.4}$$

Ez az egyenlet használható *páratlan* paritásbit generálásakor. Az A-PB regiszterláncban levő bináris szó páros paritásbitjét meghatározó egyenlet:

$$A(1)\oplus A(2)\oplus A(3)\oplus A(4)\oplus A(5)\oplus A(6)\oplus PB=0, \tag{2.5}$$

amit a következőképpen lehet átírni:

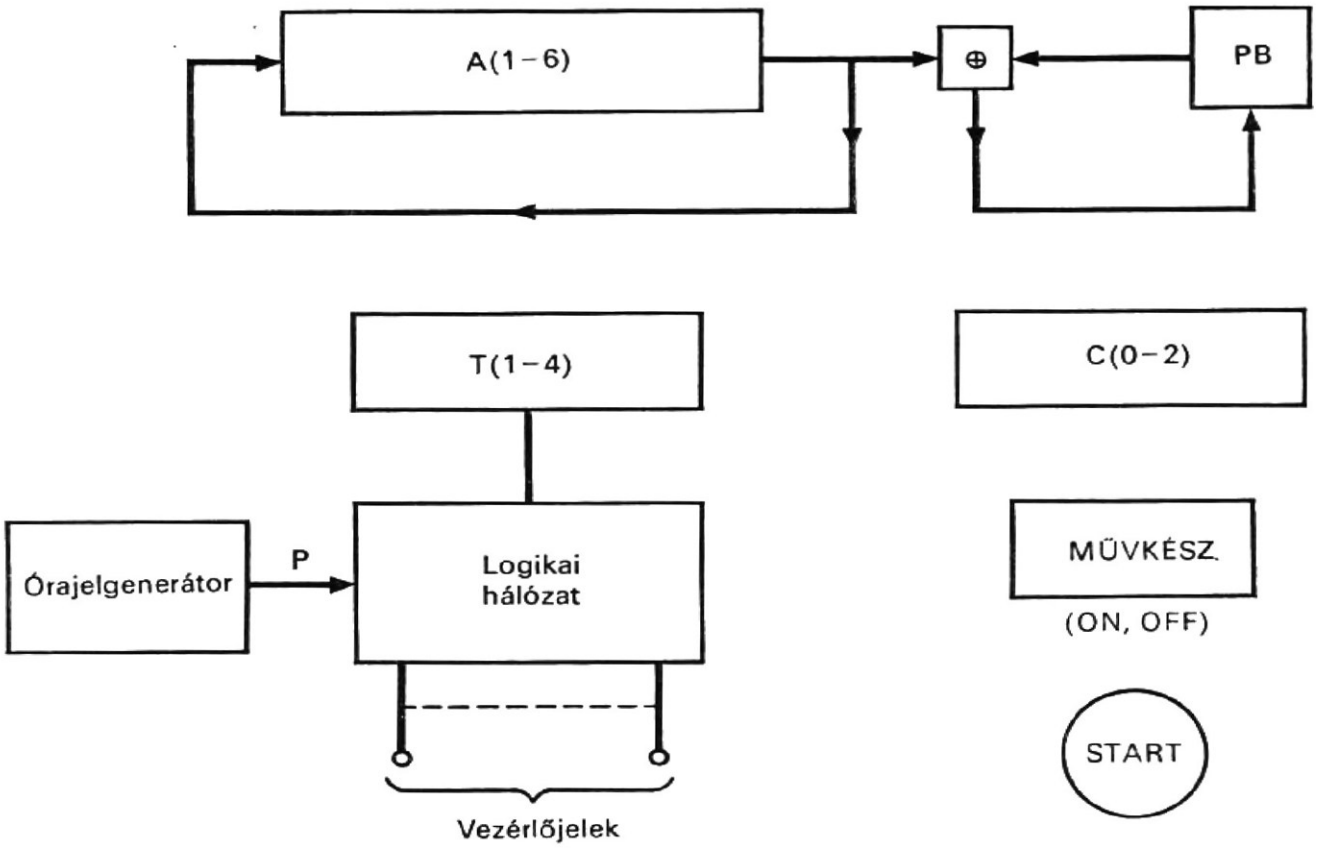
$$PB=A(1)\oplus A(2)\oplus A(3)\oplus A(4)\oplus A(5)\oplus A(6)\oplus 0 \tag{2.6}$$

Ez az egyenlet használható páros paritásbit előállításakor.

2.1.2. Logikai kapcsolási rajz

Paritásbit soros generálására használható logikai kapcsolási rajz látható a 2.1. ábrán. Az A regiszter az a léptetőregiszter, amely a bináris szót tárolja. A PB regiszterben helyezkedjék el a bináris szó paritásbitje. A C számláló számlálja az A regiszter-léptetésszintek számát, a T számláló és a P órajelgenerátor állítja elő a vezérlőjeleket. A START kapcsoló kezdeményezi a szükséges műveleteket, míg a MŰVKÉSZ indikátor a művelet befejezését jelzi. Ezeket a funkcionális elemeket a következő deklarációs utasításokkal írjuk le:

```
Comment, soros paritásgenerátor struktúrája.

Register,   A(1—6),          $léptetőregiszter
            PB,              $paritásbit-regiszter
            C(0—2),          $számláló                    (2.7)
            T(1—4),          $fázisregiszter
Switch,     START(ON),
Light,      MŰVKÉSZ(ON, OFF),
Clock       P
```

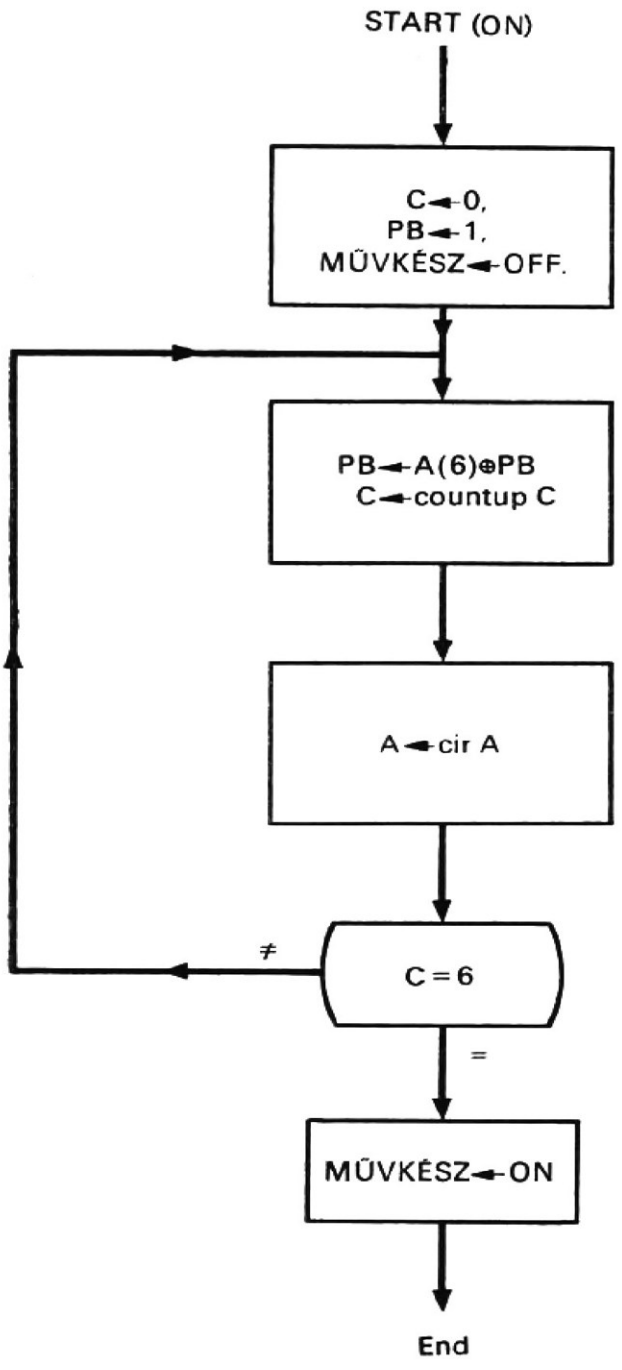


2.1. ábra
Soros paritásgenerátor tömbvázlata

2.1.3. Folyamatábra

A paritásbit előállítását a 2.2. ábrán levő folyamatábra határozza meg. Feltételezzük, hogy a bináris szó a művelet kezdetén már az A regiszterben van. Amint a 2.2. ábrán látható, ha a START kapcsoló az ON helyzetbe kerül, a C számláló 0 értéket kap és a MŰVKÉSZ indikátor OFF állapotba jut. Páratlan paritásbit generálásához a PB regisztert kezdetben 1 értékre kell állítani (kezdeti feltétel beállítása). Ezután a logikai KIZÁRÓ—VAGY művelet kerül végrehajtásra és ezzel egyidejűleg a C számláló értéke 1-gyel megnő. A következőkben az A regiszter egy bittel ciklikusan jobbra léptetendő, majd a C számláló tartalmának vizsgálata következik: elérte-e a 6-os értéket. Ha az érték még nem 6, a KIZÁRÓ—VAGY logikai művelet, a léptetés és a számlálás addig ismétlődik, amíg el nem éri a 6-os értéket. Ekkorra a kívánt paritásbit a PB regiszterben van és a MŰVKÉSZ indikátor ON állapotba kerül, jelezve ezzel a művelet befejezését.

Páros paritásbit generálásához a PB kezdetben 0 értéket kap; a többi lépés ugyanaz marad, mint az előbbiekben.



2.2. ábra
Soros paritásgenerátor folyamatábrája

2.1.4.
A műveletsorozat leírása

A sorrendi műveleteket a következő végrehajtási utasítások írják le:

Comment, itt kezdődik a soros paritásbit generálása.

```
/START(ON)/  PB←1,                                    (2.8)
             MŰVKÉSZ←OFF
             C←0,
             T←10₈,
/T(1)*P/     PB←A(6)⊕PB,
             C←countup C,
             T(1,2)←01,
```

```
/T(2)*P/     A←cir A,
             T(2,3)←01,
/T(3)*P/     IF (C=6) THEN (T(3,4)←01)
             ELSE (T(3,1)←01),
/T(4)*P)     MŰVKÉSZ←ON,
             T(4)←0,
             End
```

2.2. Soros komparátorok

Két bináris számot sorosan össze lehet hasonlítani (komparálni) annak meghatározására, hogy az egyik nagyobb-e, kisebb-e, mint a másik, vagy egyenlők-e. Az alábbiakban két előjel nélküli bináris szám és két előjeles bináris szám összehasonlítására alkalmas algoritmust és vezérlési rendszert írunk le.

2.2.1. Két bináris szám soros összehasonlítása

Az előjel nélküli bináris számok pozitív számok. Ha két pozitív számot sorosan hasonlítunk össze, akkor az összehasonlítási algoritmust a számok letapogatási iránya határozza meg. Ha a számokat bitenként jobbról balra (azaz a legkisebb helyiértékű bittől a nagyobb helyiértékű bit felé haladva) tapogatjuk le és hasonlítjuk össze, akkor ez az eljárás soros kivonással egyenértékű. Ezért az összehasonlítás eredménye mindaddig ismeretlen, amíg ismertté nem válik a legértékesebb bitektől származó átvitel értéke, vagy amíg nem találjuk egyenlőnek valamennyi összetartozó bitpárt. Ezt úgy végezhetjük, hogy a két pozitív bináris számot balról jobbra haladva tapogatjuk le és hasonlítjuk össze. Így az összehasonlítás eredményét azonnal meg lehet állapítani akkor, ha az azonos helyiértéken levő bitek nem egyenlők, mindaddig, amíg ez a két érték egyenlő, folytatjuk a letapogatást. A számok valamennyi helyiértékén el kell végezni az összehasonlítást.

Az előjeles bináris számok negatívak vagy pozitívak lehetnek. Ha két előjeles bináris számot sorosan hasonlítunk össze, akkor az összehasonlítási algoritmus nemcsak a számok letapogatási irányától, hanem attól is függ, hogyan vannak ábrázolva ezek a számok. Itt most csak előjellel és abszolút értékkel ábrázolt bináris számokat vizsgálunk. Ha a két összehasonlítandó szám előjele különböző, a pozitív előjelű szám a nagyobb. Ha a két előjel egyforma, és a számokat balról jobbra tapogatjuk le, az a szám lesz a nagyobb, amelynek az első értékes bitje nagyobb helyiértéken van, amennyiben mindkét előjel pozitív. Az első kisebb bittel (azaz 0-val) rendelkező szám lesz a nagyobb, ha mindkét előjel negatív. Ha a két előjeles számot jobbról balra tapogatjuk le, ez viszont soros kivonással egyenértékű eljárás, és az összehasonlítás eredményét mindaddig nem ismerjük, amíg a legértékesebb helyiértékű bittől származó átvitel rendelkezésre nem áll. Bármelyik irányból történjék is a számok letapogatása, a két szám csak akkor egyenlő, ha összetartozó bitjeik egyenlők.

2.2.2.
Logikai hatásvázlat előjel nélküli bináris számok összehasonlításához

Két, előjel nélküli bináris szám soros összehasonlítására alkalmas logikai kapcsolási rajz látható a 2.3. ábrán. Az A és B regiszter tárolja a két összehasonlítandó, előjel nélküli bináris számot. Az NB, EGY, és KB regiszterek az összehasonlítás eredményét jelzik. A C számláló a lépések számát számlálja. A T regiszter, a P órajelgenerátor és a K kimeneti kapcsolókkal rendelkező dekódoló generálja a vezérlőjeleket. A START kapcsoló kezdeményezi az összehasonlításhoz szükséges műveleteket. A MŰVKÉSZ indikátor jelzi az összehasonlítás befejeztét. Ezeket a funkcionális elemeket a következő deklarációs utasítások írják le:

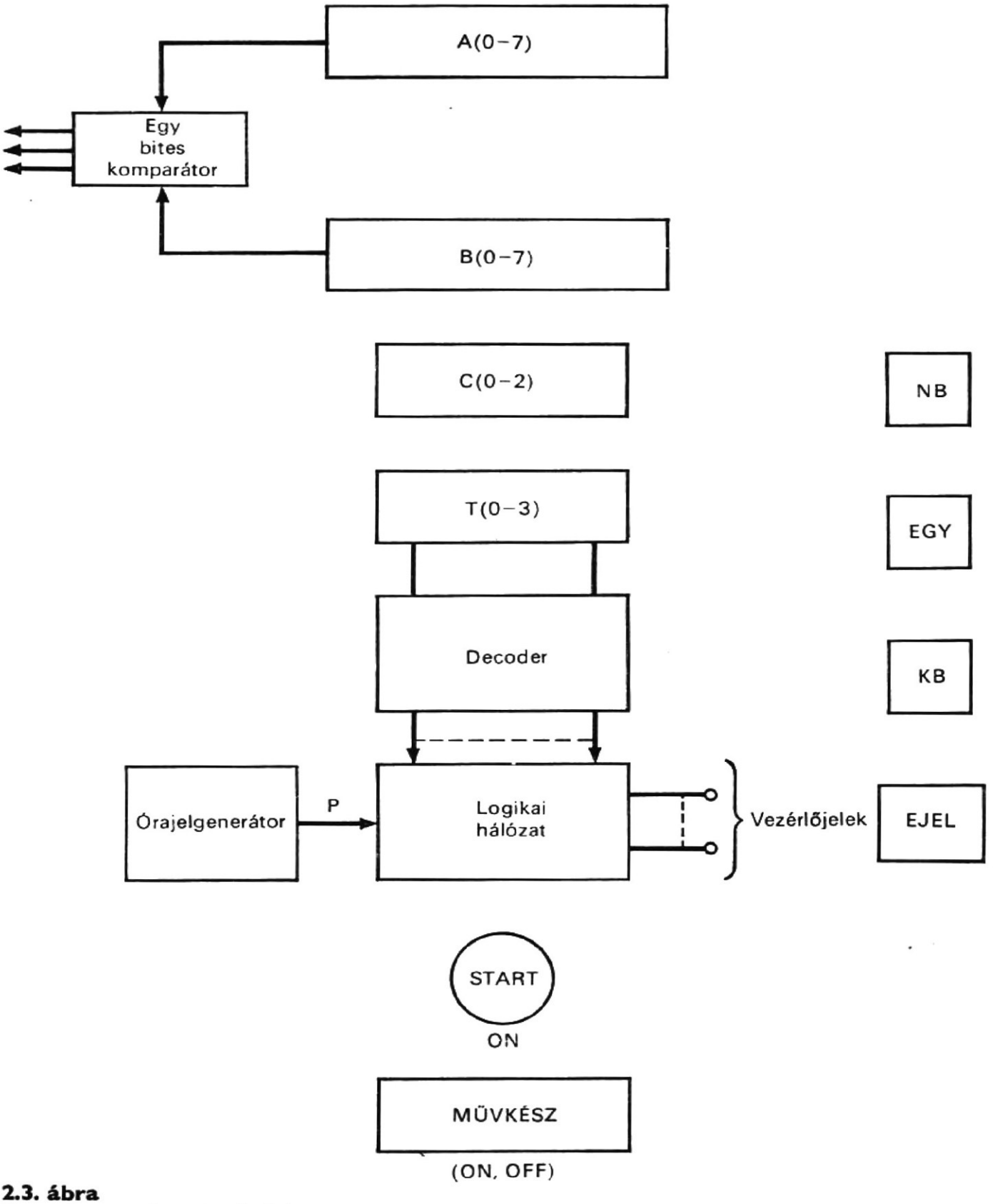
Comment, két előjel nélküli szám soros összehasonlításának leírása.

```
Register,   EGY,          $egyenlőséget jelez, ha értéke 1.
            NB,           $jelzi, hogy A nagyobb mint B, ha értéke 1.
            KB,           $jelzi, hogy A kisebb mint B, ha értéke 1.
            EJEL,         $az előjelet tárolja.
            A(0—7),       $az egyik számot tárolja. A(0) az előjelbit.
            B(0—7),       $a másik számot tárolja. B(0) az előjelbit.
            C(0—2),       $számláló.
            T(0—3),       $fázisregiszter.
Decoder,    K(0—8)=T
Switch,     START(ON)                                          (2.9)
Light,      MŰVKÉSZ.(ON, OFF)
Clock,      P
```

A 2.3. ábrán levő logikai kapcsolási rajzon látható egy háromkimenetű, egybites komparátor. Ennek három kimenete azt jelzi, hogy az A(0)—B(0) regisztersor tartalma 01, 00 vagy 11, ill. 10. Az első eset azt jelenti, hogy az A(0) bit kisebb, mint a B(0) bit, a második két eset, hogy ezek egyenlők és az utolsó eset, hogy az A(0) nagyobb B(0)-nál. Ha az A(0)—B(0) sor 01, 00 vagy 11, ill. 10 értéket tartalmaz, azok sorrendben a következő Boole-kifejezéseknek felelnek meg: $A(0)' * B(0)$, $A(0) \odot B(0)$ és $A(0) * B(0)'$. Ezeket a Boole-kifejezéseket később használjuk fel.

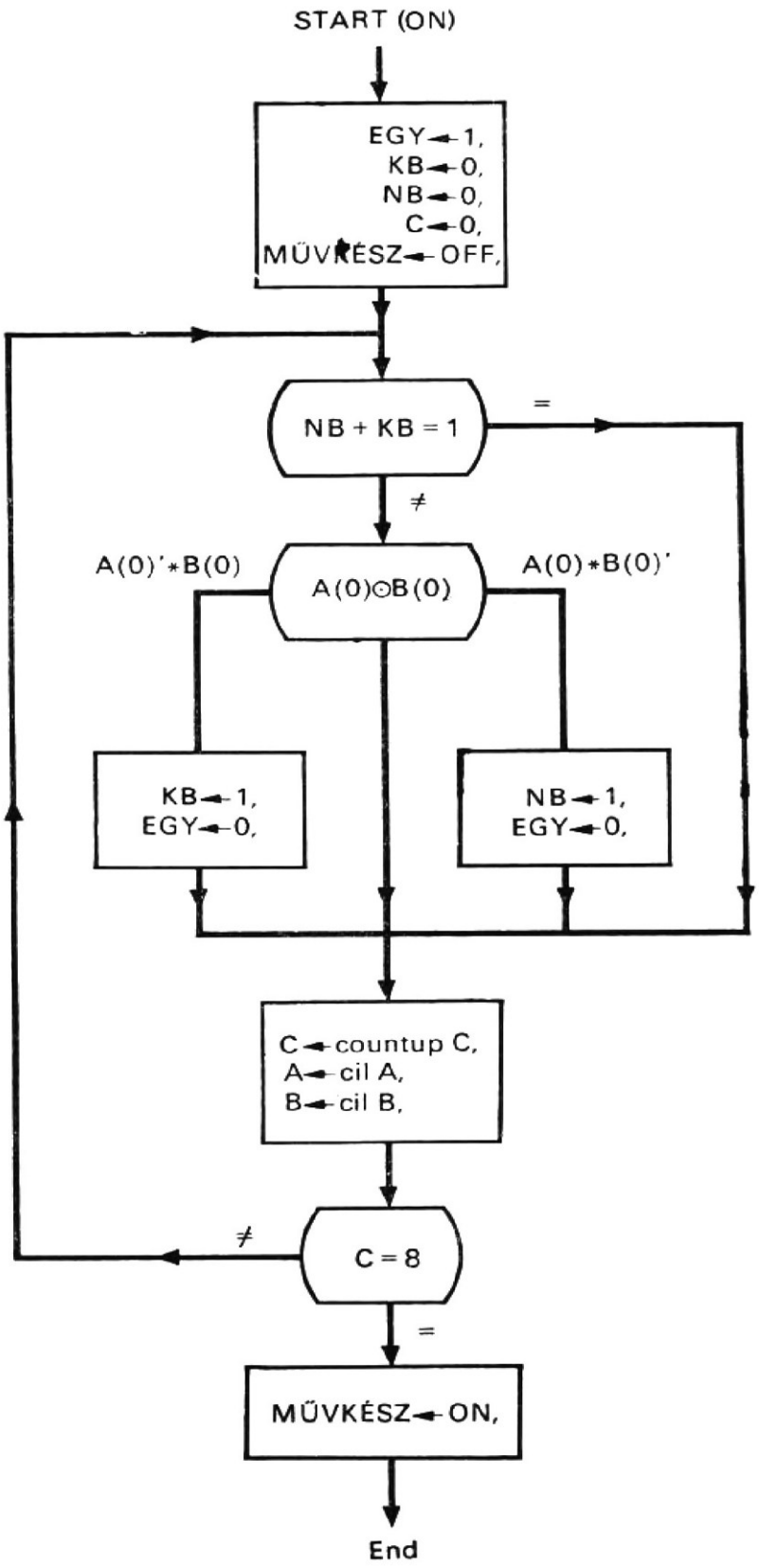
2.2.3.
Előjel nélküli bináris számok összehasonlításának folyamatábrája

Két, előjel nélküli bináris szám soros összehasonlításának folyamata látható a 2.4. ábra tömbvázlatán. Feltételezzük, hogy a két szám kezdetben az A és B regiszterben van. Ha a START kapcsoló az ON helyzetbe kerül, az EGY regiszter 1 értéket kap, a KB és NB regiszterek pedig 0-t, továbbá a C számláló is 0 értékűvé válik a MŰVKÉSZ indikátor az OFF állapotba kerül. Megvizsgáljuk az A és a B regiszter bal oldalról legelső bitjét. Ha az A(0) bit nagyobb, mint a B(0) bit, az NB regiszter 1 értéket kap és az EGY regiszter visszaállítódik 0-ra. Ha az A(0) bit kisebb, mint a B(0) bit, a KB regiszter 1 értéket kap és az EGY regiszter pedig 0-t. Semmi sem történik, ha ez a két bit egyenlő. Mindhárom esetben a C számláló értéke 1-gyel megnő, és mind az A, mind a B regisz-



2.3. ábra
Soros komparátor tömbvázlata

ter ciklikusan egy bittel balra lép. Ezután a C számláló értékét ellenőrizzük, elérte-e értéke a 8-at. Ha a C számláló értéke nem 8, az NB és KB regiszter vizsgálata következik az 1 értékre. Ha egyikük értéke 1 (vagyis az NB+KB=1), akkor az összehasonlítás elmarad. Ha egyikük sem tartalmaz 1-et, az A(0)—B(0) sor ismét ellenőrzésre kerül és újból végrehajtódik a C számláló inkrementálása, valamint az A és B regiszter léptetése. A C számlálót ismét ellenőrizzük, értéke elérte-e a 8-at. Ha a C számláló nem érte el a 8 értéket, az előbbiekben leírt ellenőrzés és mikroművelet mindaddig ismét-



2.4. ábra
Folyamatábra két, előjel nélküli bináris szám soros összehasonlításához

lődik, amíg a C számláló el nem éri a 8-at. Ekkor a MŰVKÉSZ indikátor ON állapotba kerül és az összehasonlítás befejeződött.

Megjegyezzük, hogy az előbbi algoritmus az A és a B regiszter tartalmát változatlanul hagyja az összehasonlítás végeztével.

2.2.4.
Két, előjel nélküli bináris szám összehasonlításának szimulációja

A 2.4. ábra folyamatábráján látható sorrendi műveleteket most a következő utasításokkal írhatjuk le:

```
Comment, itt kezdődik az előjel nélküli bináris számok összehasonlítása
/START(ON)/ EGY←1, KB←0, NB←0, C←0, MŰVKÉSZ←OFF, T←0,
/K(0)*P/    IF (NB+KB=1) THEN (T←2) ELSE (T←1),
/K(1)*P/    IF (A(0)—B(0)=A(0)'*B(0)) THEN (KB←1, EGY←0),          (2.10)
            T←2,
/K(2)*P/    C←countup C, A←cil A, B←cil B, T←3,
/K(3)*P/    IF (C=8) THEN (T←4) ELSE (T←0),
/K(4)*P/    MŰVKÉSZ←ON,
            END
```

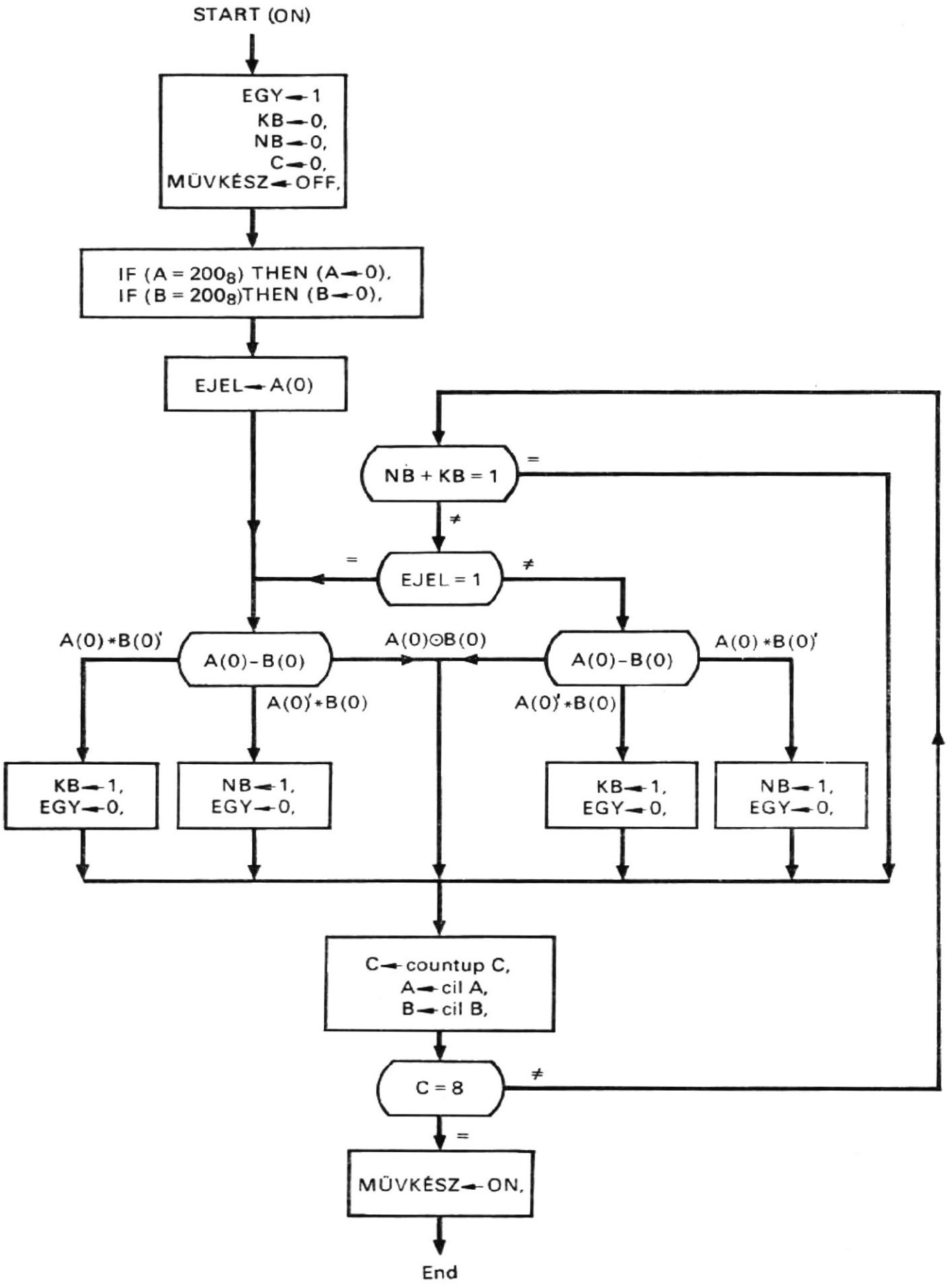
2.2.5.
Előjellel és abszolút értékkel ábrázolt bináris számok soros komparálása

A két, előjellel és abszolút értékkel ábrázolt bináris szám soros üzemű összehasonlítására szolgáló logikai hatásvázlat hasonló a 2.3. ábrán láthatóhoz és működése szintén a (2.9) utasításokkal írható le. Megjegyezzük azonban, hogy az EJEL előjel regiszter nem szerepelt a (2.10) leírásban.

A soros üzemű összehasonlítás folyamatábrája a 2.5. ábrán látható. Feltételezzük, hogy a két, előjeles szám kezdetben az A és B regiszterben van. Az előzőkhöz hasonló kezdeti feltétel beállítás után következik (ez a START kapcsolóval történik) az A és B regiszter ellenőrzése negatív nullára. Negatív nulla esetén mind A, mind B pozitív nullára állítódik. A következő lépésben az A(0)-ban tárolt előjelbit az EJEL regiszterbe kerül, későbbi felhasználás végett. Az első két, összehasonlítandó bit az előjelbit. Ebben az esetben az A(0)—B(0) regiszterláncban a 01, 00 vagy 11 és a 10 azt jelenti, hogy — megfelelő sorrendben — az A regiszterben levő szám nagyobb, egyenlő vagy kisebb, mint a B regiszterben levő szám. Ez az összehasonlítási szabály az abszolút értéket ábrázoló bitekre is érvényes, ha mindkét szám negatív. Ha mindkét szám pozitív, az abszolút értéket ábrázoló bitek összehasonlítási szabálya a következő: az A regiszterben levő szám kisebb, egyenlő, vagy nagyobb, mint a B regiszterben levő szám, ha az A(0)—B(0) regiszterlánc rendre 01, 00 vagy 11 és 10 értéket tartalmaz. A két szabály ellentétes jellege teszi szükségessé az EJEL regiszter használatát és teszi a 2.5. ábrán látható összehasonlítási algoritmust bonyolultabbá, mint a 2.4. ábrán levő. Egyébként a 2.4. és 2.5. ábra összehasonlítási algoritmusai hasonlók.

A két, előjellel és abszolút értékkel ábrázolt bináris szám soros komparálásának sorrendi műveleteit most a következő utasítások írják le:

```
Comment, itt kezdődik a két szám összehasonlítását leíró utasítás sorozata.
/START(ON)/ EGY←1, KB←0, NB←0, C←0, MŰVKÉSZ←OFF, T←0,              (2.11)
/K(0)*P/    IF (A=200₈) THEN (A←0),
            IF (B=200₈) THEN (B←0),
            T←1,
```



2.5. ábra
Folyamatábra két, előjeles, bináris szám soros összehasonlításához

```
/K(1)*P/    EJEL←A(0), T←4,
/K(2)*P/    IF (NB+KB=1) THEN (T←6) ELSE (T←3),
/K(3)*P/    IF (EJEL=1) THEN (T←4) ELSE (T←5),
/K(4)*P/    IF (A(0)−B(0)=A(0)*B(0)') THEN (KB←1, EGY←0),
            IF (A(0)−B(0)=A(0)'*B(0)) THEN (NB←1, EGY←0),
            T←6,
/K(5)*P/    IF (A(0)−B(0)=A(0)'*B(0)) THEN (KB←1, EGY←0),
            IF (A(0)−B(0)=A(0)*B(0)') THEN (NB←1, EGY←0),
            T←6,
/K(7)*P/    IF (C=8) THEN (T←8) ELSE (T←2),
/K(8)*P/    MŰVKÉSZ←ON
            END
```

2.3. A legnagyobb szám meghatározása

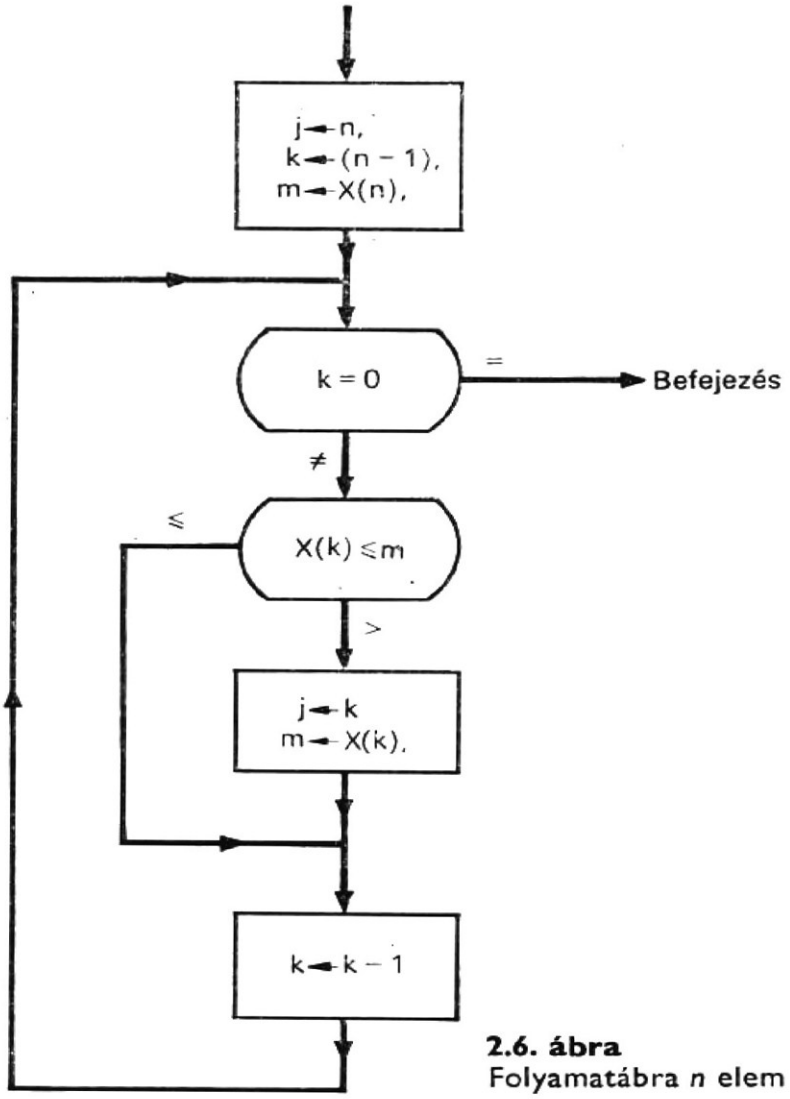
A következő feladatban megkeressük *n* darab, előjel nélküli bináris szám közül a legnagyobbat. Két változatot mutatunk be: az egyik párhuzamos, a másik soros kivonáson alapul.

2.3.1. Algoritmus a legnagyobb szám meghatározására

A 2.6. ábrán egy algoritmust láthatunk, adott $X(1),\ldots,X(n)$ bináris számok közül a legnagyobb elem meghatározására. Az ábrán *n* az elemek száma, *m* az aktuális legnagyobb elem, *k* az éppen összehasonlításra kerülő elemre mutató pointer és *j* az aktuális legnagyobb elemre mutató pointer. Amint a 2.6. ábrából látható, az első összehasonlítás az $X(n)$ és az $X(n-1)$ elem között történik, közülük a nagyobbik kerül tárolásra az *m*-ben. A következő összehasonlítás az *m* és az $X(n-2)$ között megy végbe, ezt követi az *m* és az $X(n-3)$ összehasonlítása és így tovább. Eközben mindig *m* tárolja az egyes összehasonlítások után a nagyobbik elemet. A *k* pointer $(n-1)$-től indul, és minden összehasonlítás után 1-gyel csökken. A keresési folyamat akkor ér véget, amikor a *k* eléri a 0-t.

2.3.2. A változat

Az A változat párhuzamos kivonót használ fel. Legyenek az adott bináris számok a C címregiszterrel és R tartalomregiszterrel ellátott, X tárban tárolt, előjel nélküli egészek. Legyen az X tár kapacitása 1024 szó, 24 bit szóhosszal. Tételezzük fel, hogy az *n* szám az első tárcímen van tárolva és az egészek a tár következő címein. A J és K regiszter tárolja a *j*, ill. a *k* pointert. Az A regiszter tárolja az egyes összehasonlítások utáni aktuális legnagyobb egészet. További járulékos funkcionális elem a fázisregiszter, a START kapcsoló és a MŰVKÉSZ kijelző, amint ez a 2.7. ábra hatásvázlatán látható.

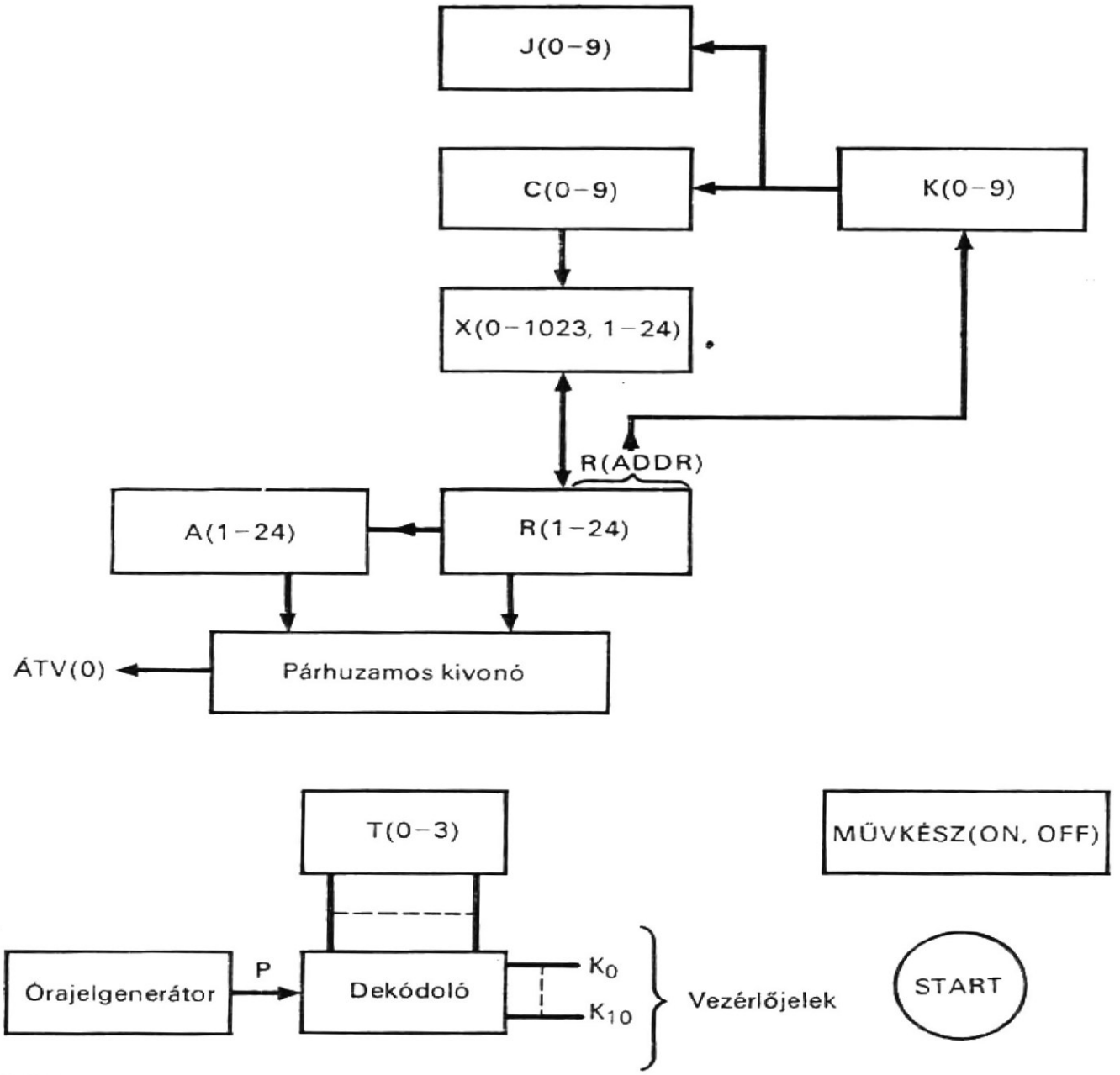


2.6. ábra
Folyamatábra *n* elem közül a legnagyobb keresésére

A legnagyobb elem keresésének egymás utáni lépéseit a 2.8. ábrán feltüntetett folyamatábra mutatja. Az indítás után, ami közben a C regiszter 0 értéket kap és a MŰVKÉSZ kijelző OFF állapotba kerül, az első szó, az *n* szám kerül kiolvasásra az X tárból. Az *n* szám azután a C tárcím-regiszterbe jut, az *n* elem utolsó elemének kiolvasása végett. Ezt az utolsó elemet az A regiszter tárolja. Ezután az utolsótól számított második elem kiolvasása következik és ez az R tartalomregiszterben tárolódik. Az A és R regiszterben levő számokat párhuzamos kivonó hasonlítja össze. Az ÁTV(0) kin.eneti bit a párhuzamos kivonó bal oldali legutolsó eleméhez tartozó átvétel. Ha az ÁTV(0) kimenet 1, ez azt jelzi, hogy az A regiszterben levő előjel nélküli bináris egész kisebb, mint az R regiszterben levő. Ebben az esetben az R regiszterből a nagyobb számot át kell vinni az A regiszterbe és az a tárcím, ahol ez a nagyobb szám van tárolva az X tárban, a J regiszterbe kerül. Ezután a következő elemet kell kiolvasni a tárból, össze kell hasonlítani az R regiszterben levő számmal. Ismét a nagyobb szám és annak tárcíme tárolódik az A, ill. a J regiszterben. Ez a folyamat mindaddig folytatódik, amíg a K el nem éri a 0-t. Ekkorra befejeződik valamennyi elem páronkénti összehasonlítása. A legnagyobb elem az A regiszterben van, a tár címe pedig a J regiszterben.

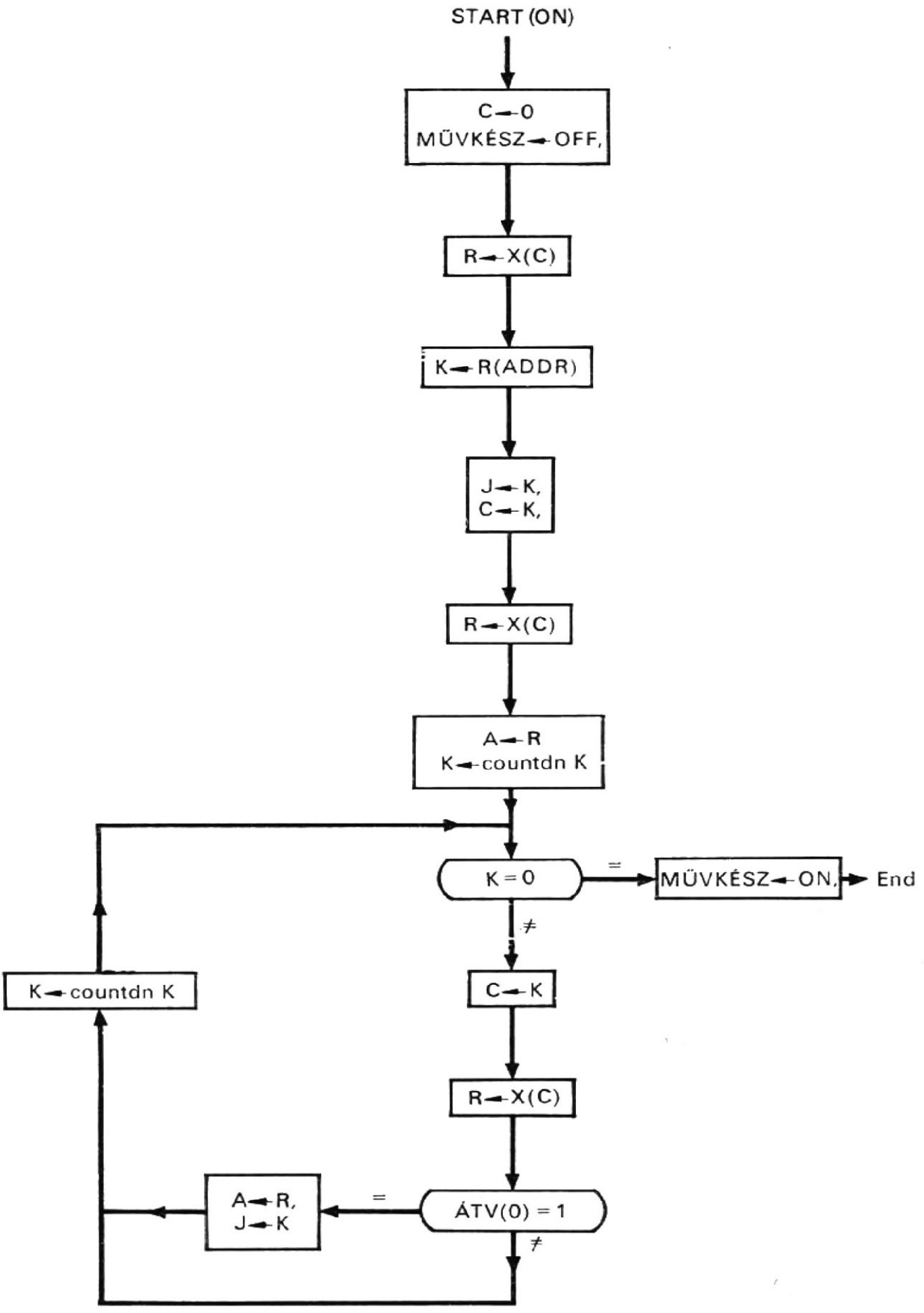
A legnagyobb egész meghatározására szolgáló előbbi struktúrát és műveletsorozatot a következő utasítások írják le:

```
Comment, legnagyobb szám meghatározása párhuzamos kivonóval
Register,   C(0—9),          $címregiszter
            R(1—24),         $tartalomregiszter
            J(0—9),          $j pointer tárolója
```



2.7. ábra
Tömbvázlat a legnagyobb elem kereséséhez, párhuzamos kivonóval

```
                K(0—9),                 $k pointer tárolója
                A(1—24),                $aktuális legnagyobb elem
                T(0—3),                 $fázisregiszter
Subregister,    R(ADDR)=R(15—24),
Memory,         X(C)=X(0—1023,1—24),
Decoder,        M(0—10)=T,
Switch,         START(ON),
Light,          MŰVKÉSZ(ON, OFF),
Terminal,       DIFF(1—24)=A(1—24)⊕R(1—24)⊕ÁTV(1—24),
                ÁTV(0—23)=A(1—24)'*R(1—24)+R(1—24)*ÁTV(1—24)+
                  +ÁTV(1—24),*A(1—24)',
                ÁTV(24)=0,

Comment, itt kezdődik az összehasonlítási műveletsorozat

/START(BE)/     C←0, MŰVKÉSZ←OFF, T←0,
/M(0)*P/        R←X(C), T←1,          $n kiolvasása
```



2.8. ábra
Folyamatábra a legnagyobb elem kereséséhez (*A* változat)

```
/M(1)*P/    K←R(ADDR), T←2,          $n tárolása K-ban
/M(2)*P/    C←K, J←K, T←3,           $n tárolása J-ben és C-ben          (2.12)
/M(3)*P/    R←X(C), T←4,             $X(n) kiolvasása
/M(4)*P/    A←R,                     $X(n) tárolása A-ban
            K←countdn K,             $ az (n−1) beállítása
            T←5,
/M(5)*P/    IF (K=0) THEN (T←10) ELSE (T←6),
/M(6)*P/    C←K, T←7,
/M(7)*P/    R←X(C), T←8,             $a következő X(C) elem kiolvasása
/M(8)*P/    IF (ÁTV(0)=1) THEN (A←R, J←K),
            T←9,
/M(9)*P/    K←countdn K, T←5,
/M(10)*P/   MŰVKÉSZ←ON,
            END
```

A (2.12)-ben a kimenetek megadásával terminalutasításokkal írtuk le a párhuzamos kivonót. A DIFF kimenő kapcsok, amelyek a párhuzamos különbség kimenetei, nincsenek felhasználva, csupán a leírás teljessége érdekében mutattuk be őket.

2.3.3. B változat

A B változat soros kivonót felhasználó megoldást mutat be. Ebben a változatban nagyrészt ugyanazok a funkcionális elemek fordulnak elő, mint amilyeneket a (2.12)-ben leírtunk, a következő eltérésekkel:

a) szerepel a BC számláló, a soros kivonás alatt végzett lépések számlálására; és

b) felhasználunk egy egybites teljes kivonót, amely a B átvételregiszterrel együtt a soros kivonást végzi. Ez a logikai hatásvázlat látható a 2.9. ábrán.

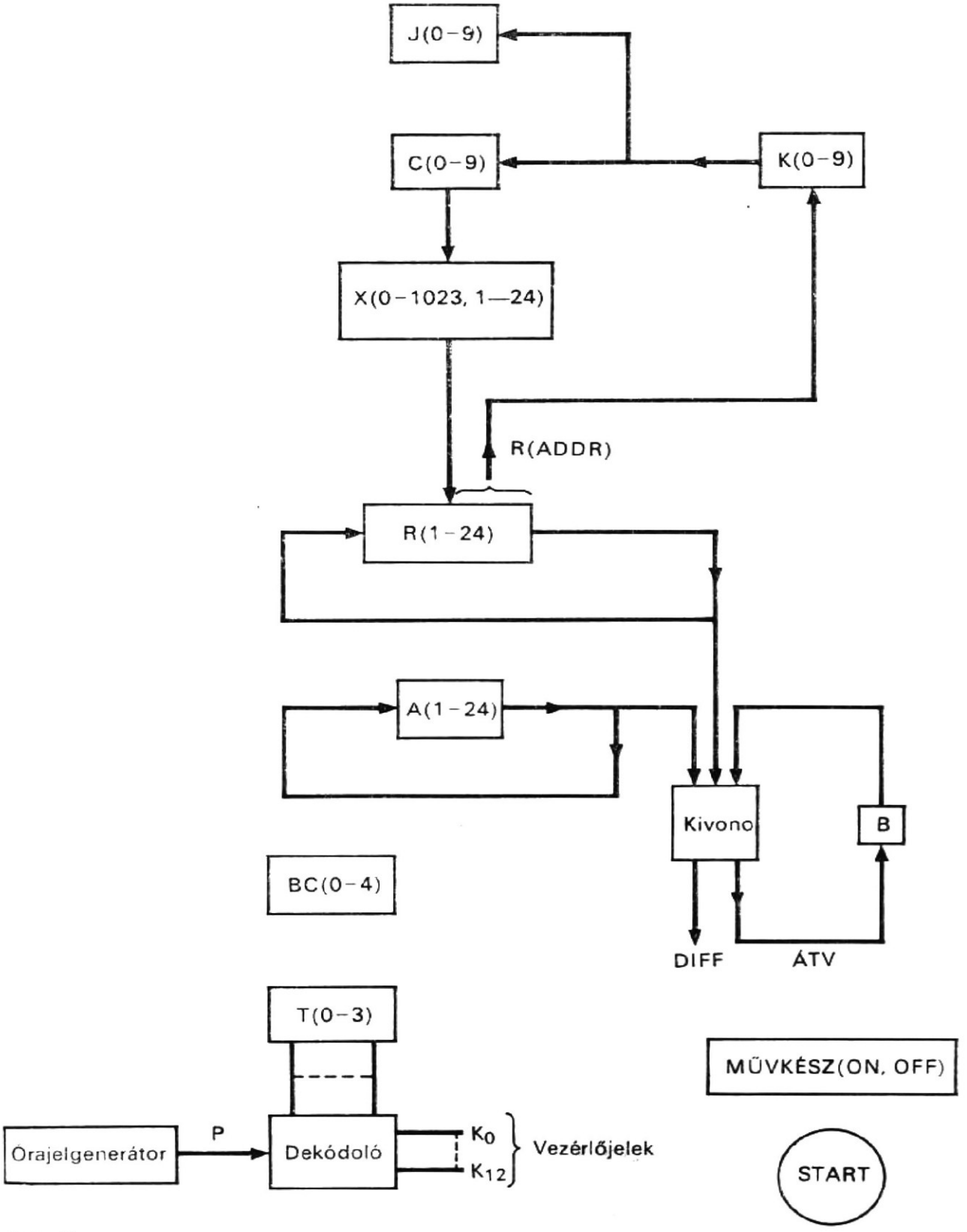
A legnagyobb egészszám meghatározásának egymás utáni lépéseit a 2.10. ábrán levő folyamatábra mutatja. A 2.8. és 2.10. ábrán levő folyamatábrák csaknem azonosak, azzal az eltéréssel, hogy a 2.10. ábrán két hurok van, míg a 2.8. ábrán csak egy. A 2.10. ábra belső hurka szolgál a két szám soros kivonására.

A B változat struktúráját és sorrendi műveleteit a következő utasítások írják le:

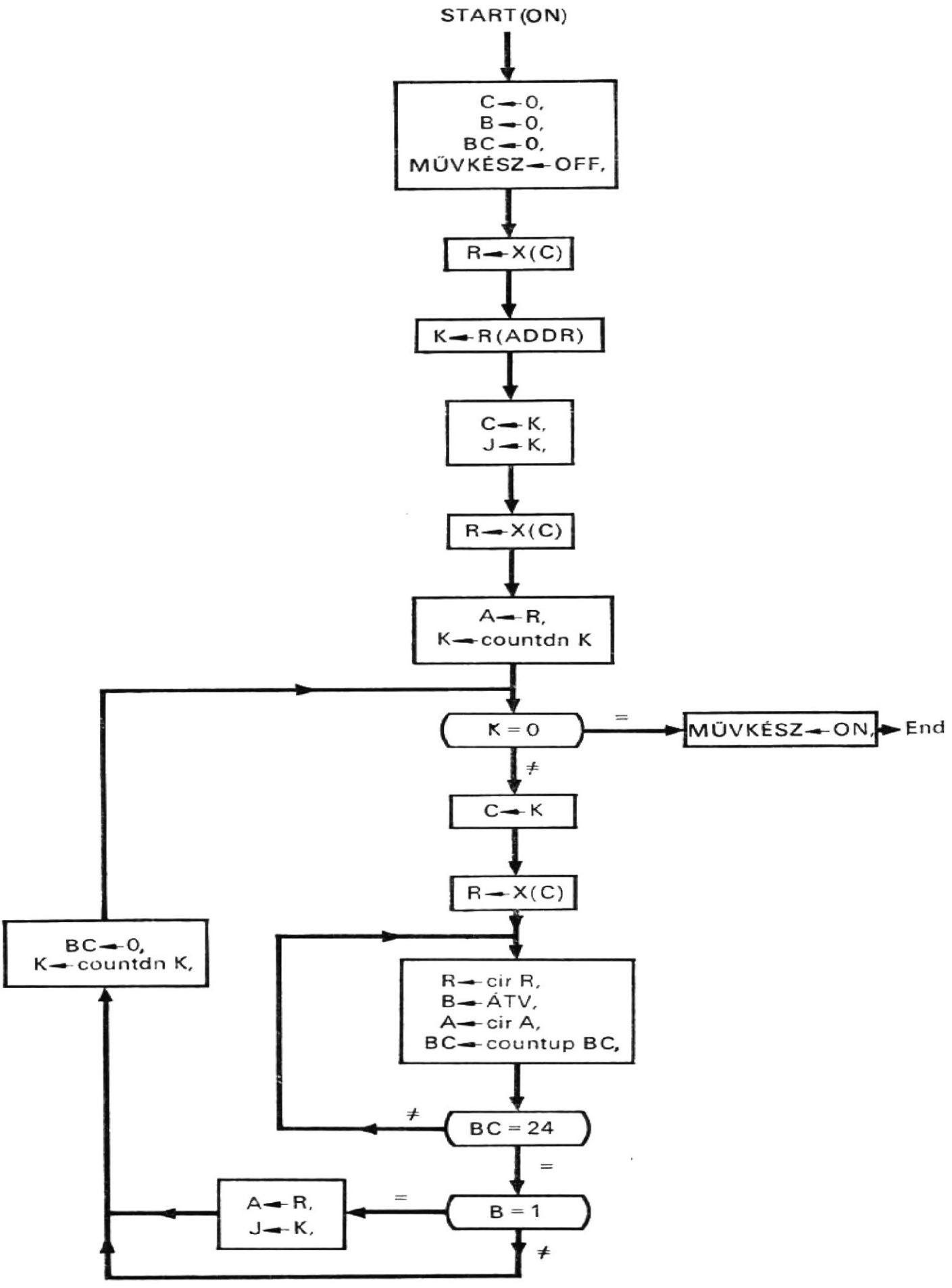
```
Comment, legnagyobb szám meghatározása soros kivonóval                  (2.13)

Register,     C(0−9),                  $címregiszter
              R(1−24),                 $tartalomregiszter
              J(0−9),                  $a j pointer tára
              K(0−9),                  $a k pointer tára
              A(1−24),                 $az aktuális legnagyobb elem tára
              B,                       $átvitelregiszter
              BC(0−4),                 $bitszámláló,
              T(0−3),                  $fázisregiszter
Subregister,  R(ADDR)=R(15−24),
Memory,       X(C)=X(0−1023,1−24),
Decoder,      M(0−12)=T,
Switch,       START(ON),
Light,        MŰVKÉSZ(ON, OFF),
Terminal,     DIFF=A(24)⊕R(24)⊕B,
              ÁTV=A(24)'*R(24)+A(24)'*B+R(24)*B
```

Comment, itt kezdődik az összehasonlítási sorozat

```
/START(ON)/  C←0, MŰVKÉSZ←OFF, B←0, BC←0, T←0,
/M(0)*P/     R←X(C), T←1,        $n kiolvasása
/M(1)*P/     K←R(ADDR), T←2,     $n tárolása K-ban
/M(2)*P/     J←K, C←K, T←3,      $n tárolása J-ben és C-ben
/M(3)*P/     R←X(C), T←4,        $X(n) kiolvasása
/M(4)*P/     A←R. T←5,           $X(n) tárolása A-ban
```



2.9. ábra
Tömbvázlat a legnagyobb elem kereséséhez, soros kivonóval



2.10. ábra
A legnagyobb elem keresésének folyamatábrája (*B* változat)

```
/M(5)*P/     IF (K=0) THEN (T←12) ELSE (T←6),
/M(6)*P/     C←K, T←7,
/M(7)*P/     R←X(C), T←8,          $a következő X(C) elem olvasása
/M(8)*P/     R←cir R, B←ÁTV, A←cir A, BC←countup BC, T←9,
/M(9)*P/     IF (BC≠24) THEN (T←8) ELSE (T←10),
/M(10)*P/    IF (B=1) THEN (A←R, J←K), T←11,
/M(11)*P/    K←countdn K, BC←0, T←5,
/M(12)*P/    MŰVKÉSZ←ON,
             END
```

2.4. Prímszámgenerátor

A prímszám 1-nél nagyobb egész szám, amely saját magán és az 1-en kívül más számmal nem osztható. Például a 2, 3, 5 és 7 prímszámok. Először bemutatunk egy, a prímszámok generálására szolgáló, könnyen programozható algoritmust, folyamatábrán. Ezután az algoritmust leírjuk a hardware realizáláshoz alkalmas formában és ily módon egy példát is adunk arra, hogyan lehet a „software leírást" „hardware leírásra" „lefordítani".

2.4.1. Prímszámok generálása

Legyen N egy lehetséges prímszám és legyen PRIM egy táblázat, amelyben a generált prímszámokat tároljuk. A PRIM táblázathoz két indexet rendelünk, ezek legyenek a J és K. A J a PRIM táblázat azon legutolsó belépési pontjára mutat, ahol egy prímszám bekerült. A K pointer végigtapogatja a PRIM táblázatot. Így a PRIM(K) a PRIM táblázat k-adik belépési pontja.

Az első két prímszámot, a 2-es és 3-as számot már kezdetben elhelyezzük a PRIM táblázat első két belépési pontján. A sorban következő többi prímszám a következő szabályok szerint generálható:

1. A lehetséges N prímszámoknak páratlan egész számoknak kell lenniük.
2. Az N-nek a PRIM(K)-val való osztása után Q hányadost és R maradékot kapunk:

$$\frac{N}{\text{PRIM}(K)} = Q + R, \tag{2.14}$$

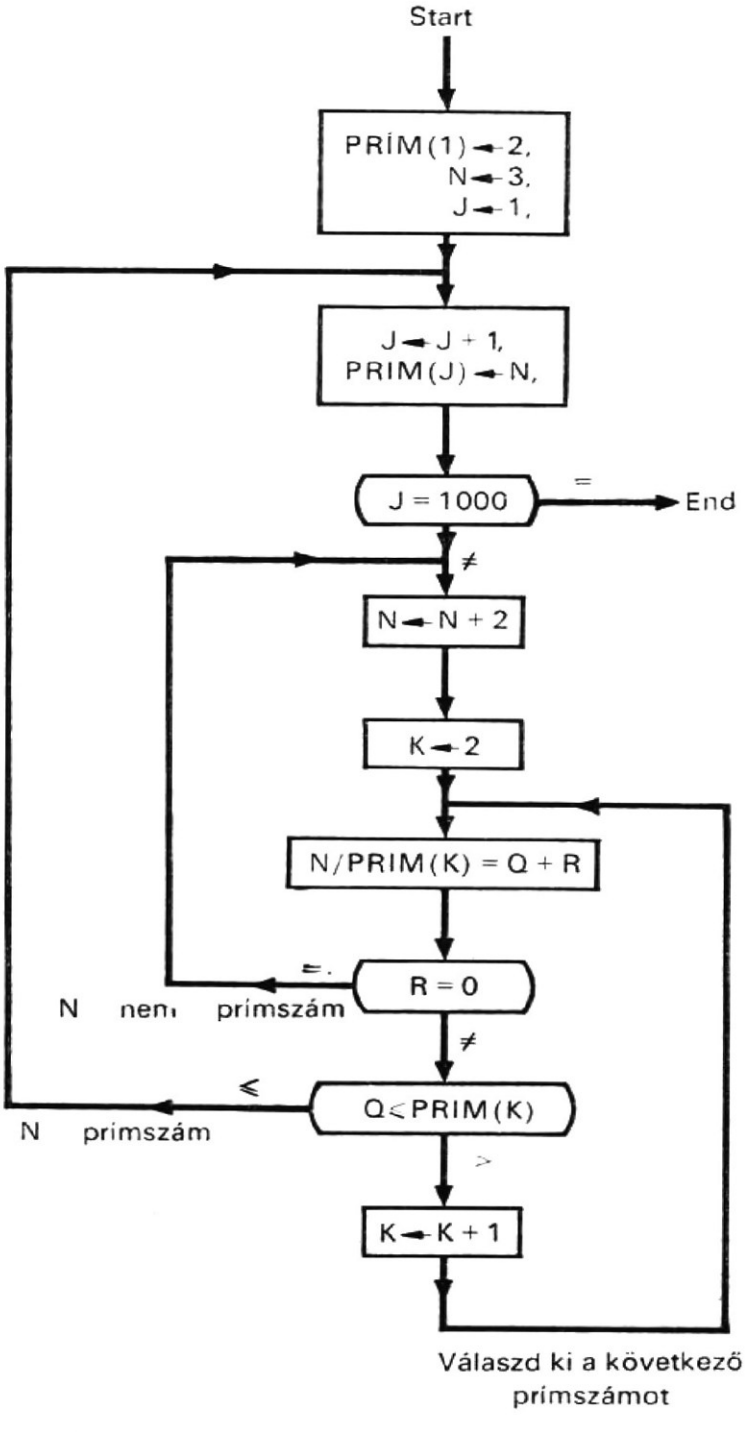
Q-t és R-et a következőképpen használjuk fel annak meghatározására, hogy vajon az N prímszám-e: Egy adott N-hez K-t 2-nek, a PRIM(K)-t 3-nak választjuk. Ezután elvégezzük a (2.14) szerinti osztást. Ha R nulla, az N nem prímszám. Ha az R nem nulla és ha a Q kisebb vagy egyenlő PRIM(K)-val, vagyis:

$$Q \leqq \text{PRIM}(K), \tag{2.15}$$

az N prímszám. Egyébként a következő, PRIM(K + 1) bemenetet választjuk a PRIM táblázatból és újból elvégezzük az előbbi osztást mindaddig, amíg vagy az R nulla nem lesz, vagy a Q kisebbé vagy egyenlővé nem válik a PRIM(K)-val.

2.4.2. Software

A 2.11. ábrán annak az algoritmusnak a folyamatábrája látható, amely az előbbiekben ismertetett szabályok szerint az első 1000 prímszámot generálja, pl. az első hat prímszámnak ezen algoritmus szerinti előállítása a 2.1. táblázatban található, ahol a Teszt feliratú oszlopok a (2.14) szerinti osztást jelölik. Az első két prímszám, a 2 és a 3 ismert. Az első *N*-nek az 5 számot választottuk és mindegyik következő *N*-et úgy kapjuk, hogy a korábbi *N*-et 2-vel növeljük, az 1. szabály értelmében. Amint a 2.1. táblázatból látható, az 5, 7, 11, 13, 17 és 19 prímszámok.



2.11. ábra
Algoritmus az első 1000 prímszám generálásához

2.1. táblázat

Az első 6 prímszám generálása

N	PRIM(K)	Teszt		Eredmény
		Q	R	
5	3	1	2/3	N prímszám, mivel $Q \leqq 3$
7	3	2	1/3	N prímszám, mivel $Q \leqq 3$
9	3	3	0	N nem prímszám, mivel $R=0$.
11	3	3	2/3	N prímszám, mivel $Q \leqq 3$
13	3	4	1/3	Folytassuk a tesztet, $Q>3$
13	5	2	3/5	N prímszám, mivel $Q \leqq 5$
15	3	5	0	N nem prímszám, mivel $R=0$
17	3	5	2/3	Folytassuk a tesztet, $Q>3$
17	5	3	2/5	N prímszám, mivel $Q \leqq 5$
19	3	6	1/3	Folytassuk a tesztet, $Q>3$
19	5	3	4/5	N prímszám, mivel $Q \leqq 5$

2.4.3. Hardware

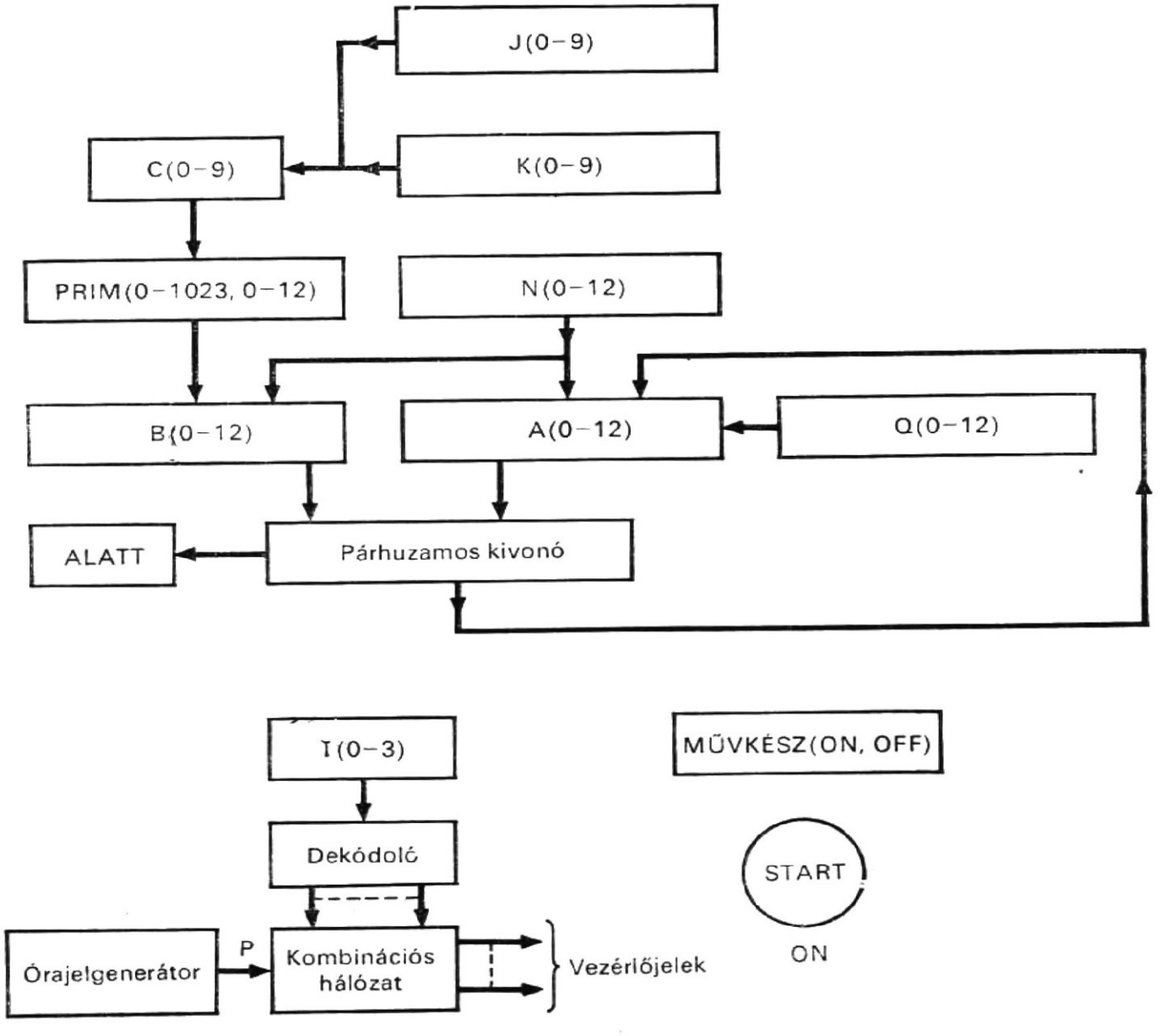
Legyen valamennyi szám bináris. Legyen PRIM az a tár, amely az előállított prímszámokat tárolja. Az 1000 prímszámhoz 1024 szó kapacitású tárat választunk. Mivel az 1000-ik prímszám 7919, a tárhoz 13 bit szóhossz szükséges. Legyen C és B a tár, cím-, ill. tartalomregisztere. A J és K regiszter a J, ill. a K indexeket tárolja. Az N számláló generálja az új számokat, és tárolja mint esetleges prímszámot. Az A regiszter akkumulátorként szolgál. A logikai hálózathoz tartozik még egy párhuzamos kivonó, amely a B regiszter tartalmát kivonja az A regiszteréből és a DIFF különbséget és az ÁTV átvételt állítja elő. Az ALATT regiszter jelzi a kivonásból származó esetleges átvételt. A START kapcsoló és a MŰVKÉSZ kijelző a műveletek elindítására, ill. befejezésére szolgál. A T regiszter, a hozzá tartozó dekódoló és a P órajelgenerátor állítja elő a vezérlőjeleket. A logikai hatásvázlat a 2.12. ábrán látható. A struktúra és működése a következő deklarációs utasításokkal írható le:

```
Comment, prímszámok generálása

Memory,     PRIM(C)=PRIM(0-1023, 0-12),
Register,   C(0-9),                  $címregiszter
            B(0-12),                 $tartalomregiszter
            J(0-9),                  $j index
            K(0-9),                  $k index
            N(0-12),                 $a lehetséges prímszám tára
            A(0-12),                 $akkumulátor                    (2.16)
            Q(0-12),                 $számláló
            ALATT,                   $a kivonás során keletkező átvételt jelzi
            T(0-3),                  $fázisregiszter
Terminal    DIFF(0-12)=A(0-12)⊕B(0-12)⊕ÁTV(0-12),
            ÁTV(0-11)=A(1-12)'*B(1-12)+B(1-12)*ÁTV(1-12)+
            +ÁTV(1-12)*A(1-12)'
            ÁTV(12)=0,
```

```
Decoder,    M(0–11)=T,
Light,      MŰVKÉSZ(ON, OFF),
Switch,     START(ON),
Clock,      P,
```

A fenti terminálutasítások a 2.3.2. pontban már szereplő párhuzamos kivonót írják le.

A 2.11. folyamatábrát megvizsgálva azt találjuk, hogy a legbonyolultabb művelet az osztás (2.14) és az egyenlőtlenségi teszt (2.15). Az osztást ismételt kivonással és számlálással lehet helyettesíteni. Ha az A regiszter 0, ez azt jelenti, hogy az R maradék 0 és az N regiszterben levő szám nem prímszám. Ha az A regiszter nem 0, a hányadost úgy kapjuk meg, hogy leszámláljuk a sikeres kivonások számát a Q regiszterben. A sikeres kivonás 1, ha a különbség nem negatív; ehhez az szükséges, hogy az A regiszter tartalma ne legyen 0 és átvétel ne forduljon elő. Ezután a Q regiszter tartalmát átvisszük az A regiszterbe, hogy a párhuzamos kivonó el tudja végezni az egyenlőtlenségi tesztet (2.15 alapján). Ha átvitel nem volt (ALATT=0), vagy az A regiszter tartalma 0 (A=0) a kivonás után, akkor az N regiszterben levő szám prímszám. Egyébként a következő prímszám kerül kiolvasásra a tárból, hogy osztóként szerepeljen a (2.14) osztásban. Ez a folyamat addig ismétlődik, amíg vagy a maradék 0, vagy amíg teljesül az egyenlőtlenségi teszt. Az előbbi eset azt jelenti, hogy az N regiszterben levő egész

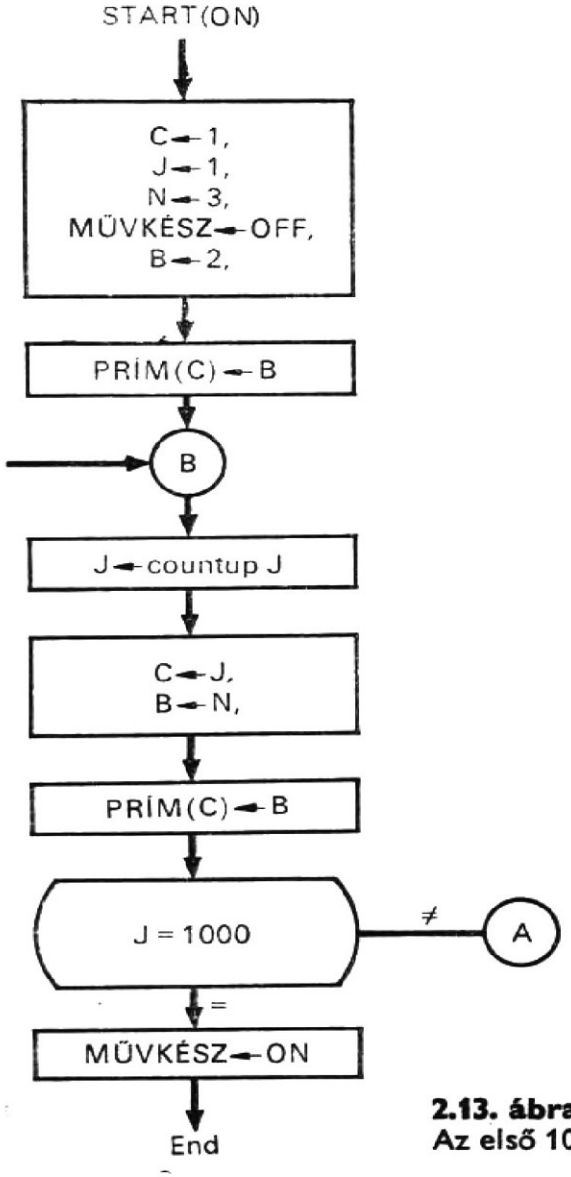


2.12. ábra
Az első 1000 prímszám generálásának logikai hatásvázlata

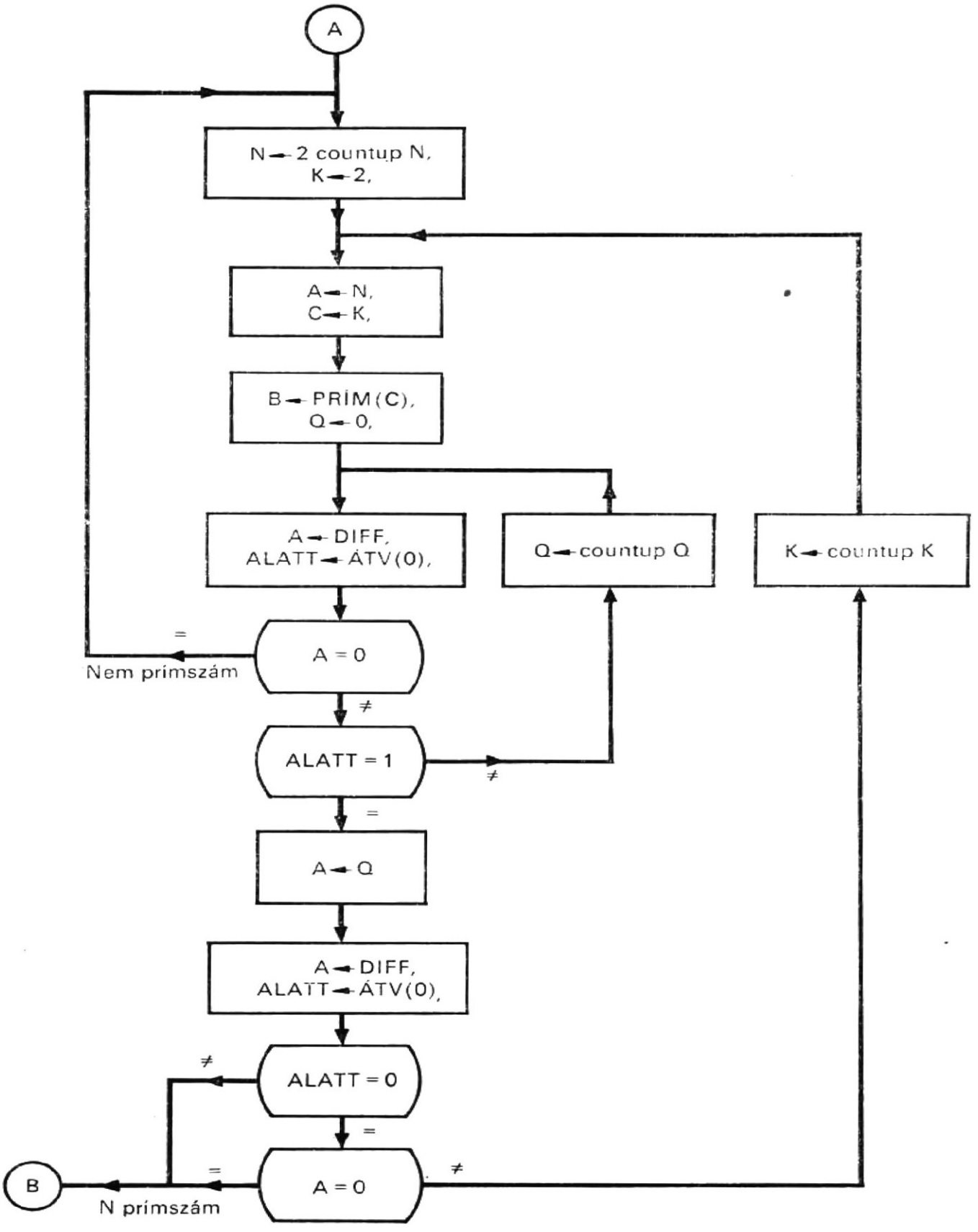
szám nem prímszám, míg az utóbbi azt, hogy prímszám. A fentiek alapján a prímszámoknak hardware úton való generálására bemutatott algoritmust a 2.13. ábrán levő sorrendi hálózat realizálja.

A 2.13. ábrán szereplő műveletsorozat a következő utasításokkal írható le:

```
Comment, itt kezdődik a generáló sorrend                                    (2.17)
/START(ON)/  C←1, J←1, N←3, B←2, MŰVKÉSZ←OFF, T←0,
/M(0)*P/     PRIM(C)←B, T←1,
/M(1)*P/     J←countup J, T←2,
/M(2)*P/     C←J, B←N, T←3,
/M(3)*P/     PRIM(C)←B, T←4,
/M(4)*P/     IF (J=1000) THEN (MŰVKÉSZ←ON) ELSE (T←5),
/M(5)*P/     N←2 countup N, K←2, T←6,
/M(6)*P/     A←N, C←K, T←7,
/M(7)*P/     B←PRIM(C), Q←0, T←8,
/M(8)*P/     A←DIFF, ALATT←ÁTV(0), T←9,
/M(9)*P/     IF (A=0) THEN (T←5) ELSE (IF(ALATT≠1)
                  THEN (Q←countup Q, T←8) ELSE (A←Q, T←10),
/M(10)*P/    A←DIFF, ALATT←ÁTV(0), T←11,
/M(11)*P/    IF (ALATT=0)*(A≠0)) THEN (K←countup K, T←6) ELSE (T←1),
             END
```



2.13. ábra
Az első 1000 prímszám generálásának folyamatábrája *(a, b)*


A
N ← 2 countup N,
K ← 2,
A ← N,
C ← K,
B ← PRÍM(C),
Q ← 0,
A ← DIFF,
ALATT ← ÁTV(0),
Q ← countup Q
K ← countup K
=
A = 0
Nem prímszám
≠
ALATT = 1
≠
=
A ← Q
A ← DIFF,
ALATT ← ÁTV(0),
≠
ALATT = 0
=
=
A = 0
≠
B
N prímszám

2.5.
Gray—bináris kódkonverter

Ha egy digitális jelet binárisan kódolunk, egy vagy több helyiértéken következik be bitváltozás (bitváltozás azt jelenti, hogy a bit értéke 0-ról 1-re vagy 1-ről 0-ra vált át), ha a számot 1-gyel növeljük. Ez bizonyos típusú analóg—digitális átalakítók esetén hibás kimenő jelet eredményezhet (pl. kódtárcsa, kódléc). Ez a nehézség elkerülhető, ha az F. Gray által 1953-ban kifejlesztett Gray-kóddal ábrázoljuk a számokat. Az alábbiakban bemutatunk egy konvertert, amely 4-bites Gray-kódot bináris kóddá alakít át.

2.5.1.
Gray—bináris kódkonverzió

A Gray-kóddal kódolt számok vagy egyszerűen a Gray-kód jellemzője [5] az, hogy egy és csakis egy bitváltozás következik be két szomszédos szám között. A 2.2. táblázatban a 0...15 decimális számok 4-bites Gray-kódját mutatjuk be; jól látható, hogy egy bit a változás bármely két szomszédos szám között.

2.2. táblázat
4-bites Gray-kód

Bináris kód				Gray-kód			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

A 2.3. táblázat a 4-bites Gray-kód, és a 4-bites bináris kód közötti kapcsolatot mutatja be, amely egyben igazságtáblázatnak is használható. Ebben a táblázatban a B_4, B_3, B_2 és B_1 — az átalakító négy kimenete — a bináris kód négy bitjét reprezentálja,

2.3. táblázat

Gray—bináris kódkonverzió igazságtáblázata

G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

2^3, 2^2, 2^1 és 2^0 súlyozással. G_4, G_3, G_2 és G_1, amelyek a Gray-kód négy bitjét reprezentálják, az átalakító négy bemenete. Az átalakító kimenetei és bemenetei közöti logikai egyenleteket azonnal megkapjuk az igazságtáblázatból. Ezek:

$$\begin{aligned} B_4 &= P_8+P_9+P_{10}+P_{11}+P_{12}+P_{13}+P_{14}+P_{15}, \\ B_3 &= P_4+P_5+P_6+P_7+P_8+P_9+P_{10}+P_{11}, \\ B_2 &= P_2+P_3+P_4+P_5+P_8+P_9+P_{14}+P_{15}, \\ B_1 &= P_1+P_2+P_4+P_7+P_8+P_{11}+P_{13}+P_{14}, \end{aligned} \tag{2.18}$$

ahol

$$\begin{aligned} P_1 &= G_4' * G_3' * G_2' * G_1, \\ P_2 &= G_4' * G_3' * G_2 * G_1, \\ P_{14} &= G_4 * G_3 * G_2 * G_1', \\ P_{15} &= G_4 * G_3 * G_2 * G_1, \end{aligned} \tag{2.19}$$

Ezeket az összefüggéseket egyszerűbben a következőképpen lehet felírni:

$$\begin{aligned} B_4 &= G_4, \\ B_3 &= G_4 \oplus G_3, \\ B_2 &= G_4 \oplus G_3 \oplus G_2, \\ B_1 &= G_4 \oplus G_3 \oplus G_2 \oplus G_1, \end{aligned} \tag{2.20}$$

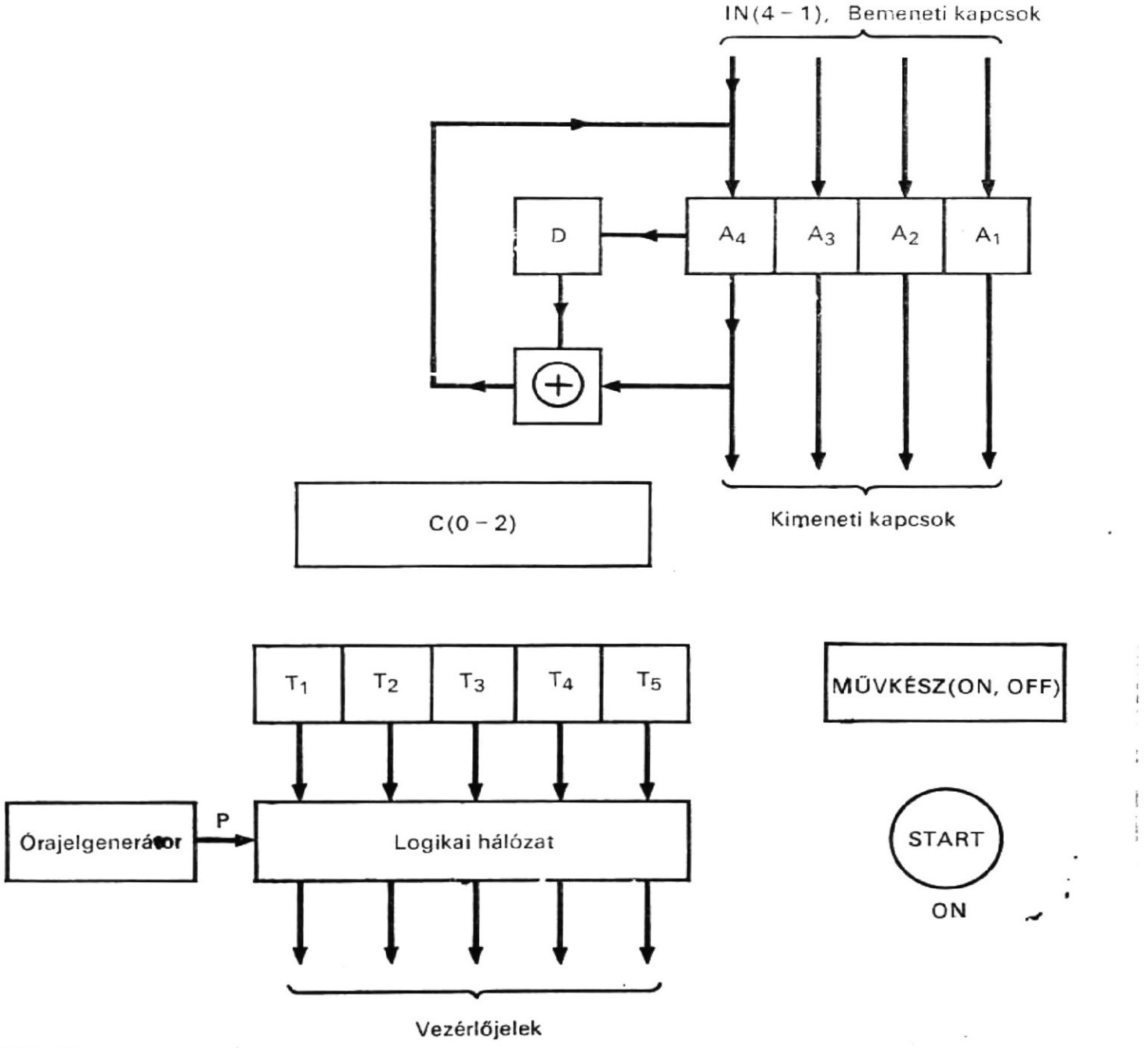
Továbbá:

$$\begin{aligned} B_4 &= 0 \oplus G_4, \\ B_3 &= B_4 \oplus B_3, \\ B_2 &= B_3 \oplus G_2, \\ B_1 &= B_2 \oplus G_1, \end{aligned} \tag{2.21}$$

A (2.21) logikai egyenletek iteratív jellege lehetővé teszi, hogy a 4-bites Gray-kódot 4-bites bináris kóddá egy egyszerű KIZÁRÓ—VAGY logikai áramkörrel alakítsuk át, soros üzemmódban. A következőkben, a soros átalakító tervezésekor, ezeket a logikai összefüggéseket használjuk fel.

2.5.2. Logikai hatásvázlat

A 4-bites Gray-kód 2.3. táblázatban levő bináris kóddá való átalakításának logikai vázlata a 2.14. ábrán látható. Az IN(4—1) bemenetek a 4-bites Gray-kód bemenetét reprezentálják. Az A regiszter egy léptetőregiszter, amely a Gray-kódot fogadja; ez tárolja az átalakítás után a bináris kódot. A D egybites regiszter átmeneti tárként szolgál. A C számláló a léptetések számát számlálja. A T regiszter és a P órajelgenerátor állítja



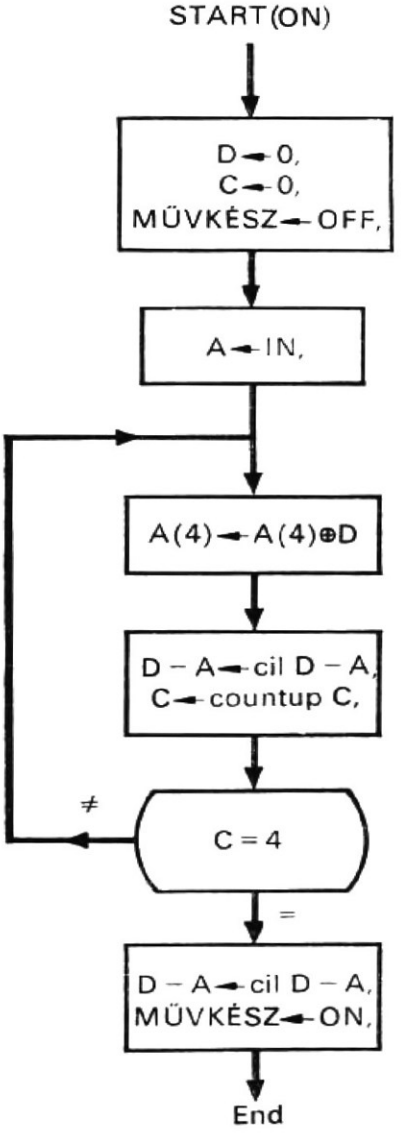
2.14. ábra
Soros Gray—bináris kódkonverter logikai hatásvázlata

elő a vezérlőjeleket. A START kapcsoló indítja a konverziót és a MŰVKÉSZ kijelző jelzi az átalakítás befejeztét. Ez a logikai struktúra a következő deklarációs utasításokkal írható le:

```
Comment, 4-bites Gray—bináris kódkonverter struktúrája
Register,   A(4—1),     $léptetőregiszter
            C(0—2),     $számláló                    (2.22)
            T(1—5),     $fázisregiszter
            D,          $átmeneti tároló
Terminal,   IN(4—1),    $Gray-kód bemenete
Switch,     START(ON),
Light,      MŰVKÉSZ(ON, OFF),
Clock       P,
```

2.5.3. Folyamatábra

Az átalakító sorrendi működését a 2.15. folyamatábra mutatja be. Amint az ábrából látható, ha a START kapcsoló a ON állásban van, a MŰVKÉSZ kijelző OFF állapotot vesz fel, és mind a C számláló, mind a D regiszter 0 értéket kap. Ezután a 4-bites Gray-kód az A regiszterbe kerül. Ezt követően az A(4)-ből és a D bitből képzett



2.15. ábra
Soros Gray—bináris kódkonverter folyamatábrája

KIZÁRÓ—VAGY művelet eredményét (vagyis $G_4 \oplus 0$-át) és A(4) tárolja, és a C számláló értéke 1-gyel megnövekszik. Ezután a D—A regiszterláncot ciklikusan léptetjük balra, egy helyiértékkel. A C számláló értékét ellenőrizzük és megvizsgáljuk, hogy elérte-e a 4-et. Ha a C értéke nem 4, megismétlődik a KIZÁRÓ—VAGY művelet, a léptetés, a számlálás és az ellenőrzés mindaddig, amíg a C értéke el nem éri a 4-et. Ekkor a D—A regiszterláncot mégegyszer balra léptetjük egy bittel, hogy az éppen megkapott bináris kód az A regiszterbe kerüljön. A MŰVKÉSZ kijelző ON állapotba kerül és az átalakítás befejeződik.

2.5.4.
A sorrendi működés leírása

A folyamatábrán látható konverziós műveletet a következő utasítások írják le:

Comment, itt kezdődik a konverzió (2.23)

```
/START(ON)/  C←0, D←0, MŰVKÉSZ←OFF, T←20₈,
/T(1)*P/     A←IN, T←cir T,
/T(2)*P/     A(4)←A(4)⊕D, T←cir T,
/T(3)*P/     D—A←cil D—A, C←countup C, T←cir T,
/T(4)*P/     IF (C=4) THEN (T←cir T) ELSE (T(2,4)←10₂),
/T(5)*P/     D—A←cil D—A, MŰVKÉSZ←ON, T←0,
             END
```

2.6.
Bináris—decimális konverter

Gyakori feladat a bináris és a decimális számok közötti konverzió. Az alábbiakban olyan konvertert mutatunk be, amely 10-bites bináris egész számot 3-számjegyes decimális számmá alakít át.

2.6.1.
Bináris—decimális konverzió

Jól ismert a bináris számot decimális számmá átalakító algoritmus [7], amelyben *n*-bites egész szám esetén az átalakítás a legnagyobb helyiértékű bitnél kezdődik. Ezt a bitet vagy megszorozzuk 2-vel és 0-t adunk hozzá, vagy megszorozzuk 2-vel, és 1-et adunk hozzá, attól függően, hogy a jobb oldalról következő bit 0 vagy 1. A kapott eredménnyel ezt ismételjük, ismét attól függően, hogy a következő bit 0 vagy 1. A kétszerezés vagy kétszerezések és 1 hozzáadásából álló folyamat $(n-1)$-szer ismétlődik, amennyiben az adott bináris számnak n bitje van.

Legyen például a bináris szám 1010110101. Az átalakítás a legértékesebb bitnél (ez jelen esetben 1) kezdődik, és a következőképpen alakul:

Lépés

1	1×2+0= 2
2	2×2+1= 5
3	5×2+0= 10
4	10×2+1= 21
5	21×2+1= 43
6	43×2+0= 86
7	86×2+1=173
8	173×2+0=346
9	346×2+1=693

Amint a fentiekből látható, az eredmény a 693-as decimális szám lesz.

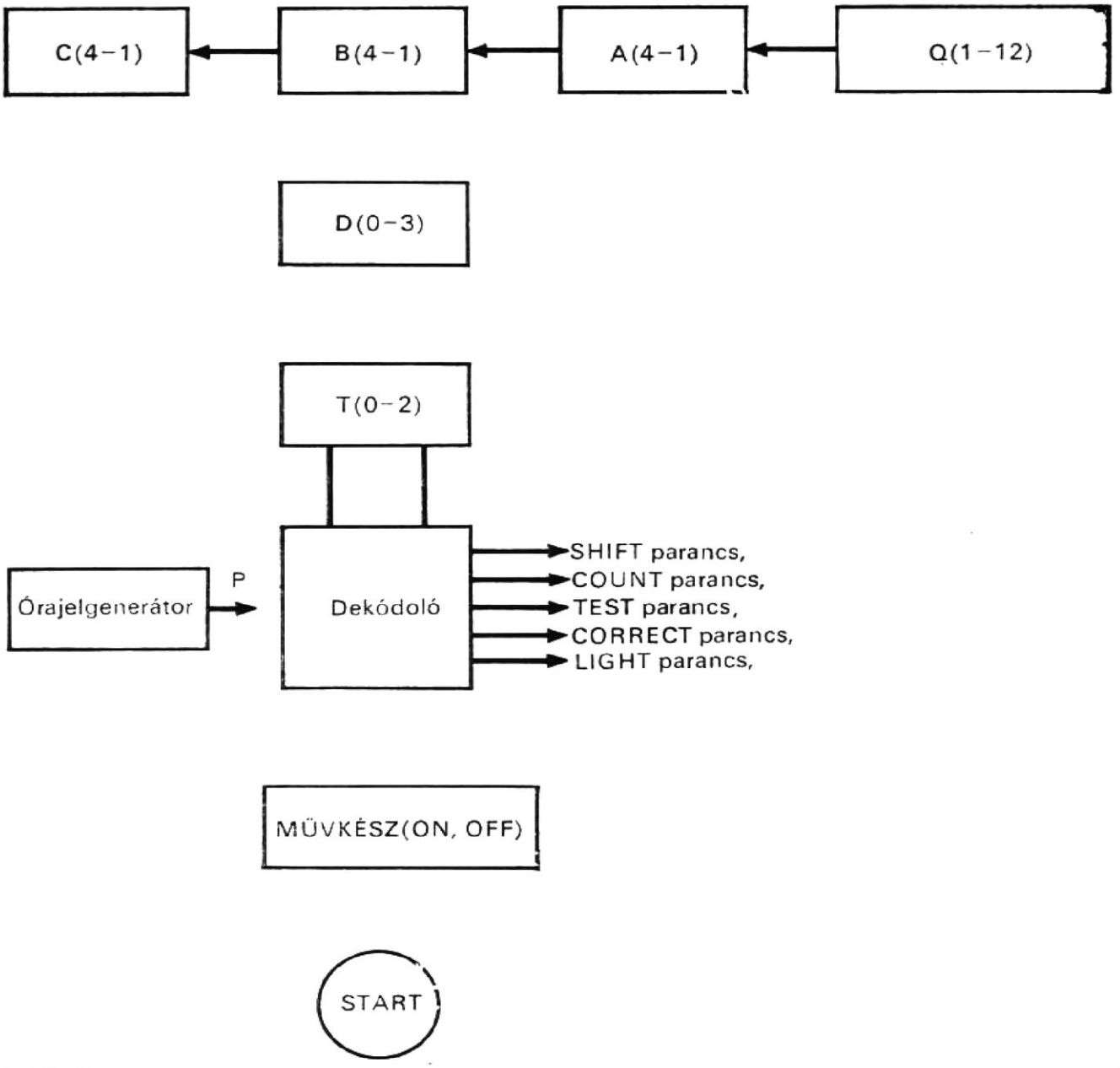
2.6.2. Logikai hatásvázlat

A bemutatásra kerülő konverter 10-bites bináris egész számot alakít át decimális számmá, amely binárisan kódolt decimális (BCD) formátumú (vagyis, minden decimális számjegy binárisan van kódolva). A konverzió három darab 4-bites regiszterben, az A, B és C regiszterekben megy végbe. Az átalakítás befejeztével az A, B és C regiszter tartalmazza a decimális szám megfelelő decimális számjegyeit, rendre 10^0, 10^1 és 10^2 súlyozással. A Q regiszter tárolja a konvertálandó 10-bites bináris számot. A D számláló a lépések számát számlálja, a START kapcsoló indítja az átalakítási és a MŰVKÉSZ kijelző jelzi a konverzió befejeztét. A T regiszter, a hozzá tartozó dekódoló és a P órajelgenerátor állítja elő a vezérlőjeleket.

A leírt elrendezést a 2.16 ábrán látható logikai hatásvázlat mutatja be és a struktúrát a következő deklarációs utasítások írják le:

```
Comment, bináris—decimális konverter struktúrája

Register,   A(4—1),              $a 10^0 súlyozású számjegy tára
            B(4—1),              $a 10^1 súlyozású számjegy tára
            C(4—1),              $a 10^2 súlyozású számjegy tára
            D(0—3),              $számláló
            Q(1—12),             $a 10-bites bináris bemenet tára      (2.24)
            T(0—2),              $fázisregiszter
Decoder,    K(0—4)=T,
Terminal,   SHIFT=K(0),          $léptetésparancs
            COUNT=K(1),          $számlálásparancs
            TEST=K(3),           $tesztparancs
            CORRECT=K(2),        $korrekciós parancs
            LIGHT=K(4),          $kijelzőt működtető parancs
Light,      MŰVKÉSZ(ON, OFF),
Switch,     START(ON),
Clock,      P,
```

A leírásra kerülő konverziós folyamathoz szükség van egy korrekciós mikroműveletre. Ennek érdekében a következő módon definiálunk egy, cor-nak nevezett speciális operátort:



2.16. ábra
Bináris—decimális kódkonverter logikai hatásvázlata

```
Operator,     X ← corX(4—1),                                    (2.25)
/begin/       IF (X=5+6+7+8+9) THEN (X ← X add 3),
              End of operator.
```

Ez az operátor az X regiszter tartalmához hozzáadja a 3-as konstansot, ha az X értéke 5, 6, 7, 8 vagy 9.

2.6.3.
Konverziós folyamat

A konverzió [4] a C—B—A—Q regiszterláncban megy végbe. Az algoritmustaz előzőkben leírtuk, amikor a 1010110101 bináris számot a 693 decimális számmá (vagyis 0110, 1001, 0011 alakba) alakítottuk át. A 2.17. ábra bemutatja a konverzió egyes lépéseit. Első lépésként a C—B—A regiszterlánc 0 értéket kap (törlés). A második lépésben a regisztersort balra egy bittel léptetjük és ily módon a legnagyobb helyiértékű bit

Lépés	Művelet	C regiszter				B regiszter				A regiszter				Bejövő bináris szám									
		C_4	C_3	C_2	C_1	B_4	B_3	B_2	B_1	A_4	A_3	A_2	A_1										
1	Értékadás	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1	0	1
2	Léptetés												1	0	1	0	1	1	0	1	0	1	
3	nincs korrekció+												1	0	1	0	1	1	0	1	0	1	
4	Léptetés											1	0	1	0	1	1	0	1	0	1		
5	nincs korrekció+											1	0	1	0	1	1	0	1	0	1		
6	Léptetés										1	0	1	0	1	1	0	1	0	1			
7	3 hozzáadása A_i-hez									1	0	0	0	0	1	1	0	1	0	1			
8	Léptetés								1	0	0	0	0	1	1	0	1	0	1				
9	nincs korrekció+								1	0	0	0	0	1	1	0	1	0	1				
10	Léptetés							1	0	0	0	0	1	1	0	1	0	1					
11	nincs korrekció+							1	0	0	0	0	1	1	0	1	0	1					
12	Léptetés						1	0	0	0	0	1	1	0	1	0	1						
13	nincs korrekció+						1	0	0	0	0	1	1	0	1	0	1						
14	Léptetés					1	0	0	0	0	1	1	0	1	0	1							
15	3 hozzáadása A_i-hez és B_i-hez					1	0	1	1	1	0	0	1	1	0	1							
16	Léptetés				1	0	1	1	1	0	0	1	1	0	1								
17	3 hozzáadása B_i-hez				1	1	0	1	0	0	0	1	1	0	1								
18	Léptetés			1	1	0	1	0	0	0	1	1	0	1									
19	3 hozzáadása A_i-hez			1	1	0	1	0	0	1	0	0	1	1									
20	Léptetés	0	1	1	0	1	0	0	1	0	0	1	1										
			6				9				3			=	1010110101								

+Nem kell korrekció, ha a bináris szám nem haladja meg a 4 (azaz, a 0100) értéket.

2.17. ábra
Példa a konverziós folyamat lépéseinek bemutatására

most bekerül az A regiszterbe. A harmadik lépésben korrekciót kell végrehajtani úgy, hogy 3-at adunk hozzá az A, B vagy C regiszterhez, ha az A, B vagy C regiszter tartalma meghaladja a 4-et (vagyis bináris kódban a 0100 értéket). Minthogy most a C, a B és az A regisztersor értéke 0000, 0000 és 0001, ebben a lépésben nincs szükség helyesbítésre. A negyedik és ötödik lépés csupán megismétli a második és harmadik lépést. Az ötödik lépésben sincs szükség korrekcióra.

A hatodik lépés végén a C, B és A regisztersor tartalma 0000, 0000 és 0101. Mivel a C regiszter tartalma meghaladja a 4-et, 0011 hozzáadásával korrekció szükséges a C regiszterben; ez a helyesbítés a hetedik lépésben megy végbe. A nyolcadik lépés folyamán a regisztersor ismét lép egyet és a C, B és A regiszterek tartalma sorrendben a következő lesz: 0000, 0001 és 0000 (vagyis a 010 BCD szám). Ha nem lett volna korrekció a hetedik lépésben, ezek a regiszterek most a 0000, 0000 és 1010 értékeket tartalmaznák, amelyek nem helyesek.

A helyesbítés további magyarázatot igényel. Ha egy bináris szám 10-es értékű az A, B vagy C regiszterben, akkor a 6 konstanst kellene hozzáadni a regiszter tartalmához, hogy helyesen generálódjék a decimális átvitel, mivel a decimális számok BCD számok alakjában vannak ábrázolva. Ezt a korrekciót azonban a balra léptetés előtt lehet alkalmazni; ebben az esetben a korrekció módosul, és 3-at adunk hozzá a regiszterhez 6 helyett.

A 14-ik lépés befejeztével az A, B és C regiszter sorra a következő értékeket tartalmazza: 0000, 1000 és 0110. Tehát, a B és C regiszter tartalma 8 és 6. Minthogy mindkettő meghaladja a 4-et, mindegyikhez 3 hozzáadódik a 14-ik lépésben. Balra léptetés művelet következik ismét a 16-ik lépésben, és a C, B és A regiszterek tartalma 0001, 0111 és 0011, ami a 173 decimális számot reprezentálja.

Az előbbiekben leírt balra léptetés és korrekció (vagy nemkorrekció) lépések addig ismétlődnek, amíg valamennyi, a Q regiszterben levő bit be nem lép az A regiszterbe. Ekkor a regisztersor tartalma 0110, 1001, 0011, ami a helyes eredményt, a 693 számot reprezentálja.

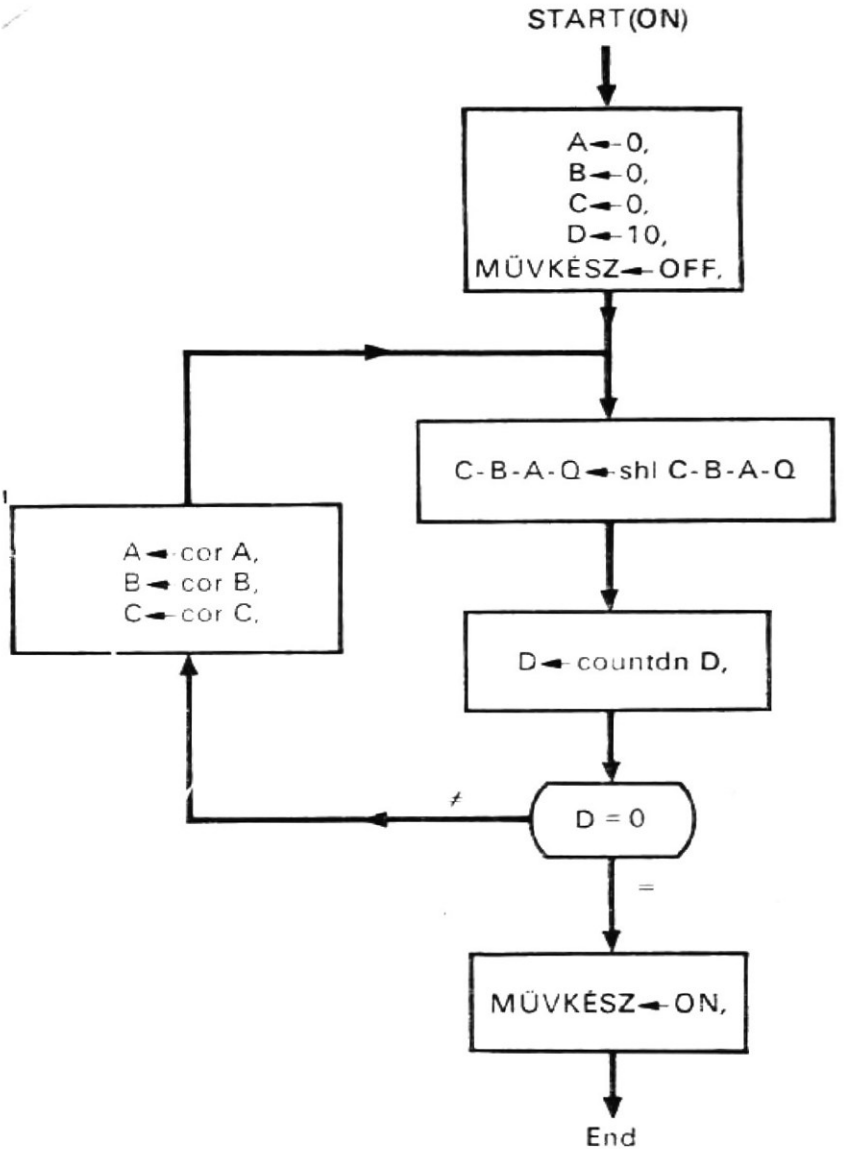
2.6.4.
A műveletek sorrendjének leírása

A fentiekben meghatározott konverziós folyamatot a 2.18. folyamatábrával lehet leírni. A folyamatábrán látható hurokban négy mikroművelet kerül végrehajtásra, mégpedig: SHIFT, COUNT, TEST, CORRECT, vagyis balra léptetés, számlálás, teszt és korrekció. Feltételezzük, hogy a bináris szám kezdetben a Q regiszterben van. A műveletek sorrendje most a következő utasításokkal írható le:

Comment, itt kezdődik a konverzióleíró utasítássorozat (2.26)

```
/START(ON)/     A←0, B←0, C←0, D←0, MŰVKÉSZ←OFF, T←0,
/SHIFT*P/       C—B—A—Q←cil  C—B—A—Q, T←1,
/COUNT*P/       D←count D, T←3,
/TEST*P/        IF (D=0) THEN (T←4) ELSE (T←2),
/CORRECT*P/     A←cor A, B←cor B, C←cor C, T←0,
/LIGHT*P/       MŰVKÉSZ←ON,
                END
```

A cor operátor a (2.25) utasításban definiált speciális operátor.



2.18. ábra
Bináris—decimális konverter folyamatábrája

2.7. Tárolt átvitelű összeadás

Két, előjel nélküli bináris szám párhuzamos összeadását speciális párhuzamos összeadóval Mercer [2] javasolta. Az alábbiaknak a Mercer-összeadást és annak kiegészítését mutatjuk be, nullatesztre és egy összehasonlítási tesztre.

2.7.1. A Mercer-összeadási algoritmus

A félösszeadónak két bemenete (*X* és *Y*), és két kimenete van: CARRY az átvitel és SUM az összeg. A SUM az *X* és *Y* közötti KIZÁRÓ—VAGY művelet eredménye, a CARRY-átvitel az *X* és *Y* közötti ÉS műveletből jön létre. Legyenek az összeadandók *n*-bites pozitív bináris számok. A Mercer-algoritmus a párhuzamos összeadóval végzett lépések sorozatából áll, az alkalmazott párhuzamos összeadónak *n* félösszeadója van (teljes összeadók helyett). Az első lépés során a két összeadandó bit szimultán adódik össze a félösszeadók *n* szakaszán; ez az összeadás *n* összegbitet és *n* átvitelbitet eredményez. Mind az összegbiteket, mind az átvitelbiteket tárolják, és

azután a második lépésben adják őket össze; ez az összeadás megint *n* összegbitet és *n* átvitelbitet eredményez. Ismét tárolják az összegbiteket és az átvitelbiteket, és a következő lépésben összeadják őket. Ez az összeadási folyamat mindaddig folytatódik, amíg valamennyi átvitel bit 0 nem lesz; ekkor befejeződik az összeadás és az összeg a kért válasz.

n-bites számok esetén a Mercer-algoritmussal végzett összeadáshoz szükséges maximális idő az *n* lépés ideje, azonban az átlagos összeadási idő ennél sokkal kisebb. Burks és mások [1] arról számolnak be, hogy az átlagos összeadási idő csak 4,6 lépés ideje, ha az *n* 40. Mercer úgy véli, hogy az algoritmusához szükséges átlagos összeadási idő esetleg még kisebb, mivel a 4,6 lépés átlagos ideje véletlenszerűen kiválasztott, teljes hosszúságú számokhoz kellett, míg ténylegesen az összeadások egy része egyszerű és rövid. Továbbá, a csupán félösszeadókat tartalmazó párhuzamos összeadóval végzett összeadás egyes lépéseihez szükséges idő azért is rövidebb, mert az átviteleket minden egyes összeadás után tárolják, ahelyett, hogy engednék őket továbbterjedni. A Mercer-összeadó aszinkron, mivel a két szám összeadásának befejezéséhez szükséges időt nem ismerjük előre.

2.7.2. Utasításleírás

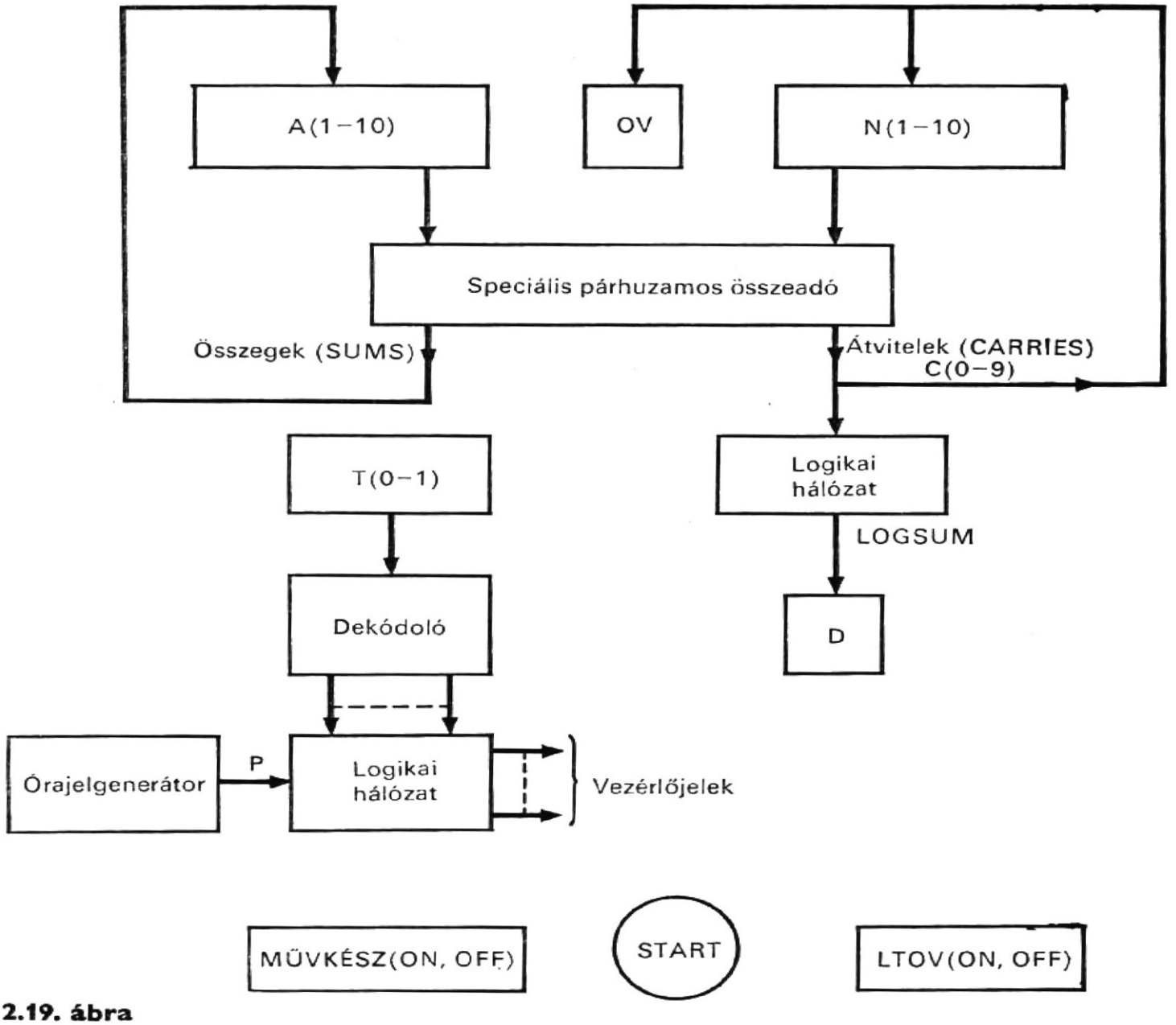
Legyen a két, előjel nélküli szám 10-bites és kezdetben legyenek az A és az N regiszterben tárolva. Az OV túlcsordulás-indikátor tárolja a párhuzamos összeadó legértékesebb szakaszából származó átvitelt az összeadás során. Speciális logikai hálózat gondoskodik a LOGSUM-nak nevezett kimenő adat kiadásáról. Ez a kimenet jelzi egy vagy több átvitel jelenlétét a félösszeadók bármelyik szakaszán, kivéve a legértékesebb helyet. A LTOV és a MŰVKÉSZ kijelzők a túlcsordulást, ill. az összeadás befejeztét jelzik. A T regiszter, a hozzá tartozó dekódoló és a P órajelgenerátor állítja elő a vezérlőjeleket. A START kapcsoló indítja az összeadásokat. Ezek a funkcionális elemek a 2.19. ábrán levő logikai hatásvázlaton láthatók.

A tárolt átvitelű összeadás műveleteinek sorrendjét a 2.20. folyamatábra mutatja. Amint látható, a START kapcsolóval végzett indítás után az A és N regiszter megfelelő bitjeinek logikai összege (vagyis a bitek közötti logikai KIZÁRÓ—VAGY művelet eredménye) beíródik az A regiszterbe; ezek a félösszeadókból jövő összegbitek. Az összeadás átvitelbitjei az N és a túlcsordulás regiszterekbe kerülnek; ezek a félösszeadók kimenetbitjei. Ugyanakkor, a LOGSUM kimenetén megjelenő jel beíródik a D regiszterbe, amit 0-ra ellenőriznek. Ha a D regiszter 0-t tartalmaz, ezt azt jelenti, hogy nem keletkezett átvitel az utolsó összeadási lépésben. Ezután megvizsgáljuk az OV regisztert. Ha 1-et tartalmaz, a LTOV túlcsordulás-indikátor ON állapotba kerül. Egyébként pedig a MŰVKÉSZ kijelző kerül az ON állapotba, hogy jelezze az összeadás befejeztét.

A Mercer-féle tárolt átvitelű összeadás struktúráját és a műveletek végrehajtási sorrendjét a következő utasítások írják le:

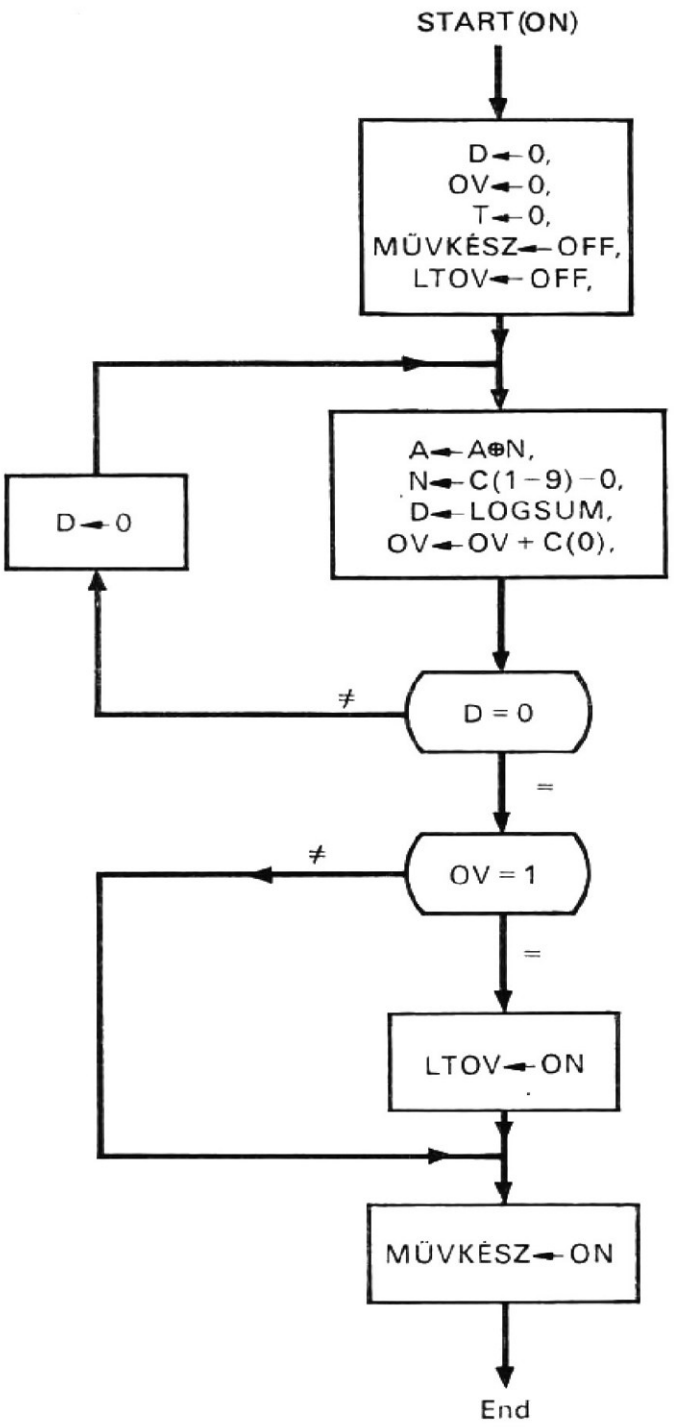
Comment, a tárolt átvitelű összeadó struktúrája és utasítássorozata (2.27)

```
Register,   A(1—10),     $összeadandó regisztere
            N(1—10),     $összeadó- vagy átvitelregisztere
            T(0—1),      $fázisregiszter
            D,           $átvitelindikátor
            OV,          $túlcsordulás-indikátor
Switch,     START(ON),   $start-kapcsoló
Light,      LTOV(ON, OFF) $túlcsordulás-indikátor
```



2.19. ábra
A Mercer-féle többlépéses összeadás logikai hatásvázlata

```
                MŰVKÉSZ(ON, OFF), $a művelet befejezésének indikátora
Terminal,       C(0—9)=A(1—10)*N(1—10),
                LOGSUM=C(1)+C(2)+C(3)+C(4)+C(5)+C(6)+C(7)+
                   +C(8)+C(9),
Decoder,        K(0—3)=T,
Clock,          P,

Comment, itt kezdődik az összeadást megvalósító utasítások sorozata

/START(ON)/     D←0, OV←0, T←0, MŰVKÉSZ←OFF, LTOV←OFF,
/K(0)*P/        A←A⊕N,
                N←C(1—9)—0,
                OV←OV+C(0),
                D←LOGSUM, T←1,
/K(1)*P/        IF (D=1) THEN (D←0, T←0)
                ELSE (IF(OV=1) THEN (T←2) ELSE (T←3)),
/K(2)*P/        LTOV←ON, T←3,
/K(3)*P/        MŰVKÉSZ←ON,
                END
```



2.20. ábra
A Mercer-féle többlépéses összeadás folyamatábrája

2.7.3. Nullára ellenőrzés

Az előbbi leírást tovább kiterjeszthetjük egy vizsgálattal, amely azt ellenőrzi, hogy egy adott szám nulla-e. Ez az ellenőrzés azon a meggondoláson alapul, hogy egy számhoz önmagát hozzáadva legalább egy átvitel lesz, kivéve, ha a szám nulla.

Ez a leírás ugyanazt a logikai hatásvázlatot használja, mint az előző, kivéve, hogy még egy ZERO regiszterre is szükség van. Az ellenőrzendő szám az A regiszterben helyezkedik el. Az A regiszter tartalmát az N regiszterbe viszik át. Ezt követően végrehajtásra kerül az összeadás és a LOGSUM kimenetéről jövő jelet a D regiszterbe írják. Ha D regiszter nullát tartalmaz, az A regiszterben levő szám 0 és a ZERO regiszter 1 értékre állítódik, annak jelzésére, hogy az eredmény nulla, és a ellenőrzés befejeződött. A nullára történő ellenőrzés utasítássorozatának leírása az alábbiakban látható:

Comment, itt kezdődik a nullára ellenőrzés utasítássorozata (2.28)

```
Register,       ZERO
/START(ON)/     D←0, T←0,
/K(0)*P/        N←A, T←countup T,
/K(1)*P/        D←LOGSUM, T←countup T,
/K(2)*P/        IF (D=0) THEN (ZERO←1), T←countup T,
                END
```

2.7.4.
Az összehasonlítási teszt leírása

A tárolt átvitelű összeadást ki lehet egészíteni egyenlőségi vizsgálattal is, amely azt ellenőrzi, hogy két tag egyenlő-e. Ez a teszt azon alapul, hogy két bináris szám közötti bitenkénti logikai KIZÁRÓ—VAGY eredménye akkor és csakis akkor nulla, ha a két szám egyforma.

Ez a leírás ugyanazt a logikai hatásvázlatot használja, mint a nullára történő ellenőrzés. A két, előjel nélküli bináris számot az A és az N regiszter tárolja. A két szám összehasonlításához a bitenként KIZÁRÓ—VAGY műveletük eredményét a speciális párhuzamos összeadó szolgáltatja és az eredmény az A regiszterben tárolódik. Ezután nullára ellenőrzik az A regiszter tartalmát. Az összehasonlítási vizsgálathoz szükséges utasítássorozat leírása a következő:

Comment, itt kezdődik az összehasonlítási tesztsorozat (2.29)

```
Register,       ZERO
Decoder,        K(0—4)=T,
/START(ON)/     D←0, MŰVKÉSZ←OFF, T←0,
/K(0)*P/        A←A⊕N, T←countup T,
/K(1)*P/        N←A, T←countup T,
/K(2)*P/        D←LOGSUM, T←countup T,
/K(3)*P/        IF (D=0) THEN (ZERO←1), T←countup T,
/K(4)*P/        MŰVKÉSZ←ON,
                END
```

2.8.
Tekeeredmény-számláló számítógép

A tízbábus tekézés népszerű sport és jól ismert az eredmény kiszámításához használható algoritmus. Egy olyan számítógépet mutatunk be, amely valamely játékos pontszámát számítja ki, abban a sorrendben, ahogy az egyes gurítások ponteredményei beérkeznek.

2.8.1.
A tekézés játékszabályai

A tízbábus tekejátékban a bábukat egy háromszögű alakzat mentén állítják fel a tekepálya egyik végében, és egy golyót gurítanak feléjük a másik végéből, azzal a céllal, hogy a lehető legtöbb bábut üssék le. Egy vagy két golyó gurítása egy *menetet* tesz ki. Minden egyes menet után újból felállítjuk a tíz bábut. Egy *játszma* tíz menetből áll.

Egy-egy gurítás pontszámát a ledöntött bábuk száma adja meg. Ha egy játékos mind a tíz bábut ledönti az első golyóval, azt *telitalálatnak* nevezik. Egyetlen ilyen gurítás kitesz egy menetet. A telitalálat pontértéke a gurítás pontszáma (azaz 10), amihez hozzájön a következő két gurítás pontszáma. Ha a játékos mind a tíz bábut két gurítással dönti le, ezt *tartaléknak* nevezik; ez a két golyóval való gurítás egy menetet tesz ki. A tartalék pontértéke a két gurítás összege (azaz 10), plusz még egy következő gurítás pontszáma. Ha a játékos nem üti le két golyóval mind a tíz bábut, ez a két golyóval való gurítás akkor is egy menetet tesz ki. Ekkor a pontszám a két gurítás eredményének összege (azaz 0 és 9 közötti érték), plusz pontszám nélkül. Az egyes menetekben két golyónál többet nem lehet gurítani. A telitalálat vagy tartalék előfordulására a menetben mint a menet *státuszára* hivatkoznak.

Minden egyes gurításhoz tartozik egy gurításpontérték. Mindegyik menetnek van egy menetponteredménye. A menetponteredményt a menetben levő gurítások pontszámaiból és az esetleges telitalálat vagy tartalék prémiumaiból számítják. Miután egy golyót elgurítottak, azt a ponteredményt, amelyet a golyópontszámból és a menetstátuszból lehet kiszámítani, *összegzett pontszámnak* nevezik. Egy játszma ponteredménye a tíz menet pontszámának összege; ez az összegzett pontszám a játszma végén. A játszma akkor végződik, amikor elegendő golyót gurítottak a tizedik menet pontszámainak kiszámításához. Mivel a telitalálat vagy a tartalék plusz számát a következő menet vagy menetek golyópontszámai határozzák meg, ha szükséges, egy vagy két külön (extra) golyót kell gurítani abban az esetben, ha ezek az utolsó menetben fordultak elő. Egy játszmában gurított golyók minimális száma 12, míg a maximum 21. A minimális pontszám 0, ami akkor fordul elő, ha egyetlen golyó sem döntött le egyetlen bábut sem, míg a maximális pontszám 300, ami akkor jön létre, ha minden golyó min-

2.4. táblázat
Tekejáték meneteinek pontszámai

Menet	Gurításpontszám		Menetstátusz	Menetpontszám		
	1-ső	2-ik		nincs prémium	prémium	prémiummal
1	9	1	tartalék	10	10	20
2	10	—	telitalálat	10	0+9	19
3	0	9	két gurítás	9	—	9
4	10	—	telitalálat	10	3+7	20
5	3	7	tartalék	10	3	13
6	3	5	két gurítás	8	—	8
7	10	—	telitalálat	10	10+4	24
8	10	—	telitalálat	10	4+5	19
9	4	5	két gurítás	9	—	9
10	8	1	két gurítás	9	—	9
Végső ponteredmény						150

den esetben tíz bábut üt le. A játékot két vagy több játékos játssza. Akié a legnagyobb végső ponteredmény, az a győztes.

A tekejáték ponteredményének számítására a 2.4. táblázatban láthatunk példát. Kényelem kedvéért a tekejáték pontszámát néha *végső ponteredménynek* nevezik. A 2.4. táblázatban a következők találhatók: a golyópontszámok, a menetstátusz, a menetpontszám prémiummal és anélkül és a végső ponteredmény. Amint látjuk, négy telitalálat és két tartalék fordul elő. A végső ponteredmény 150.

2.8.2.
Tekeponteredmény számítása

A tekejáték ponteredményének számítására az egyik mód az, hogy kiszámítjuk minden egyes menet pontszámát és azután összeadjuk a tíz menetben elért pontszámot, így megkapjuk a végső ponteredményt. A 2.4. táblázatban találunk erre példát, ahol feltüntettük a menetpontszámokat a plusz pontszámmal és anélkül, és külön magukat a plusz pontszámokat is. A menetpontszám kiszámítását az aktuális menet státusza és a következő menet befolyásolják. Számunkra hatféle menetstátusz lehet érdekes:

1. két gurítás;
2. tartalék;
3. telitalálat;
4. tartalék, amelyet telitalálat követ;
5. telitalálat, amelyet tartalék követ;
6. telitalálat, amelyet telitalálat követ.

Megjegyezzük, hogy az olyan menet-ponteredményének kiszámításához, amelyben telitalálat vagy tartalék fordul elő, szükség van a következő menet (vagy menetek) gurításpontszámára (vagy pontszámaira). Egy játékos aktuális menetének kiszámításához ki kell számítani minden bejövő golyó összegzett ponteredményét. A végső ponteredmény az összegzett gurításeredmény a játszma végén. A 2.5. táblázatban egy olyan példa található, ahol a gurításpontszámok (és így a végső ponteredmény is) ugyanazok, mint a 2.4. táblázatban. Amint látható, adott gurításpontszám esetén az összegzett ponteredményhez a gurításpontszám egy, két vagy három összeadása szükséges, a menetstátusztól függően. Például egy összeadás elegendő egy kétgolyós menet első vagy második gurításpontszámához (l. a 15. és 16. számú gurítást a 2.5. táblázatban). Viszont két összeadás kell az első gurításpontszámhoz, ha az előző menet státusza tartalék (l. a 7., 8. és 9. számú golyókat); két összeadás szükséges a kétgurításos pontszám mindegyikéhez, ha az előző menet státusza telitalálat (l. a 3., 4. és 5. számú gurításokat). Hasonló a helyzet, ha a telitalálatot tartalék követi (l. a 6., 7. és 8. számú gurításokat), vagy, ha a tartalék után telitalálatot sikerült elérni (l. az 1...5. számú gurításokat). Abban az esetben, ha a telitalálatot ismét telitalálat követ (l. a 11. és 12. számú gurításokat), a következő menet első gurításának pontszámában három összeadásra van szükség.

Az összegzett ponteredmény végelszámolásához meg kell határozni, hogy nulla, egy vagy két extra gurításra van-e szükség. A 10-ik menet extra gurításaira mutat példákat a 2.6. táblázat, ahol hatféle menetstátusz szerepel. Amint látható, nincs szükség extra gurításra, ha a menetstátusz kétgurításos. Egy extra gurítás kell, ha a státusz tartalék és két extra gurítás szükséges a maradék négy esetben. Ezen extra gurítás pontszámait egyszer hozzá kell adni az összegzett ponteredményhez, kivéve a telitalálat—telitalálat eset első gurításának pontszámát; ezt a gurításpontszámot kétszer kell hozzáadni.

2.5. táblázat

Tekejáték összegezett pontszámai

Gurításszám	Gurítás pontszáma	Menetszám	Menetstátusz	Összegezett pontszám *a) b) c) d) e)*
1	9	1		0+ 9+—+—= 9
2	1	1	tartalék	9+ 1+—+—= 10
3	10	2	telitalálat	10+10+10+—= 30
4	0	3		30+ 0+ 0+—= 30
5	9	3	két gurítás	30+ 9+ 9+—= 48
6	10	4	telitalálat	48+10+—+—= 58
7	3	5		58+ 3+ 3+—= 64
8	7	5	tartalék	64+ 7+ 7+—= 78
9	3	6		78+ 3+ 3+—= 84
10	5	6	két gurítás	84+ 5+—+—= 89
11	10	7	telitalálat	89+10+—+—= 99
12	10	8	telitalálat	99+10+10+—=119
13	4	9		119+ 4+ 4+ 4=131
14	5	9	két gurítás	131+ 5+—+ 5=141
15	8	10		141+ 8+—+—=149
16	1	10	két gurítás	149+ 1+—+—=150

Megjegyzés:
a) Az összegezett pontszám kezdeti értéke.
b) Aktuális gurításpontszám.
c) Prémium telitalálat vagy tartalék miatt.
d) Prémium telitalálat—telitalálat miatt.
e) Az összegezett pontszám új értéke.

2.6. táblázat

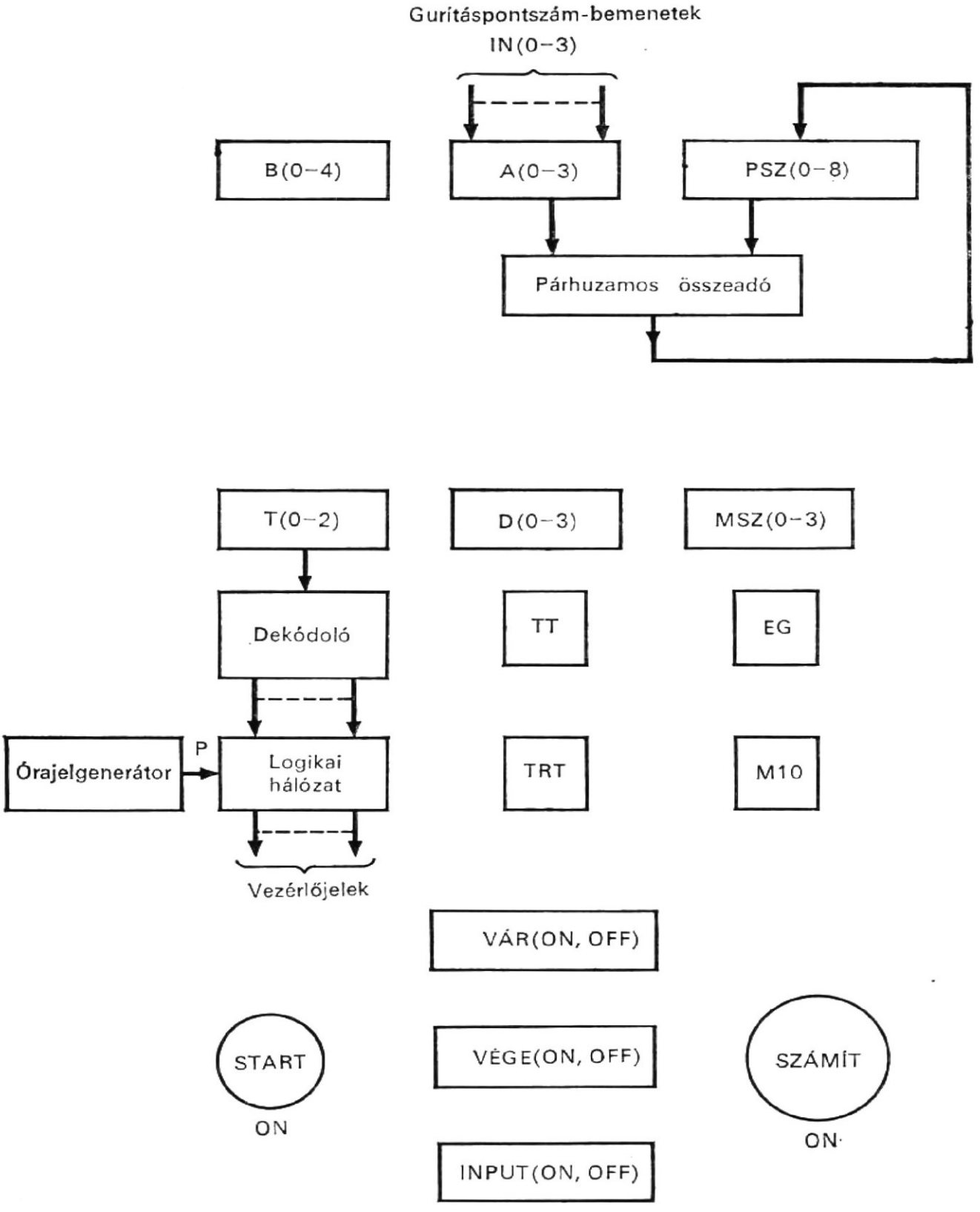
Példák extra gurításra a 10-ik menethez

Menetstátusz	Gurításpontszámok		Extra gurítás
	9-ik menet	10-ik menet	
Két gurítás	—	3,5	
Tartalék	—	4,6	3
Telitalálat	—	10	3,5
Tartalék—telitalálat	4,6	10	3,5
Telitalálat—tartalék	10	4,6	3,5
Telitalálat—telitalálat	10	10	3,*5

* Ezt a gurításpontszámot kétszer kell hozzáadni.

2.8.3. Logikai hatásvázlat

A tekejáték pontszámát meghatározó számítógép logikai hatásvázlata a 2.21. ábrán látható. Ez a hatásvázlat nem tartalmaz elemeket ahhoz, hogy a játékos manuálisan bevigye az egyes gurításpontszámokat. Feltételezzük azt, hogy ha a gurításpontszám rendelkezésre áll az IN(0—3) bemeneti kapcsokon az INPUT jelzőlámpa az ON állapotba kerül.



2.21. ábra
Tekeeredmény-jelző számítógép logikai hatásvázlata

Az A regiszter a bemeneti regiszter: ez fogadja az egyes gurításpontszámokat. A B regiszter a gurítás számának, az MSZ a menetek számának a számlálója. Ha egy menet két gurításból áll, az első gurítás pontszámát ideiglenesen a D regiszter tárolja és a második gurítás pontszáma az A regiszterben tárolódik. A párhuzamos összeadó összeadást végez el oly módon, hogy veszi a PSZ pontszámú regiszter és az A regiszterekből az operandusokat és az eredményt a PSZ regiszterben tárolja. A PSZ regiszter tárolja az összegzett ponteredményt és így ez tartalmazza a végső pontszámot is a számítás befejeztekor.

Ha 1 az állapota, a TT regiszter telitalálatot, a TRT regiszter tartalékot jelez; az EG regiszter az első gurítást és az M10 az utolsó menetet indikálja. Amikor ON állapotba kerül, a VÁR kijelző azt mutatja, hogy a számítógép várja a következő gurításpontszámot, az INPUT jelzőlámpa pedig azt, hogy rendelkezésre áll a következő gurításpontszám, míg a VÉGE jelzőlámpa a játszma végét jelzi. A START kapcsoló indítja a számítógép működését. A SZÁMÍT kapcsoló a számítás kezdetére ad parancsot, továbbá a számítás folytatására a következő gurításpontszámra való várakozás után. A T regiszter, a K végértékű dekódoló és a P órajelgenerátor állítja elő a vezérlőjeleket.

Az előbbi logikai hatásvázlatot a következő utasítások írják le:

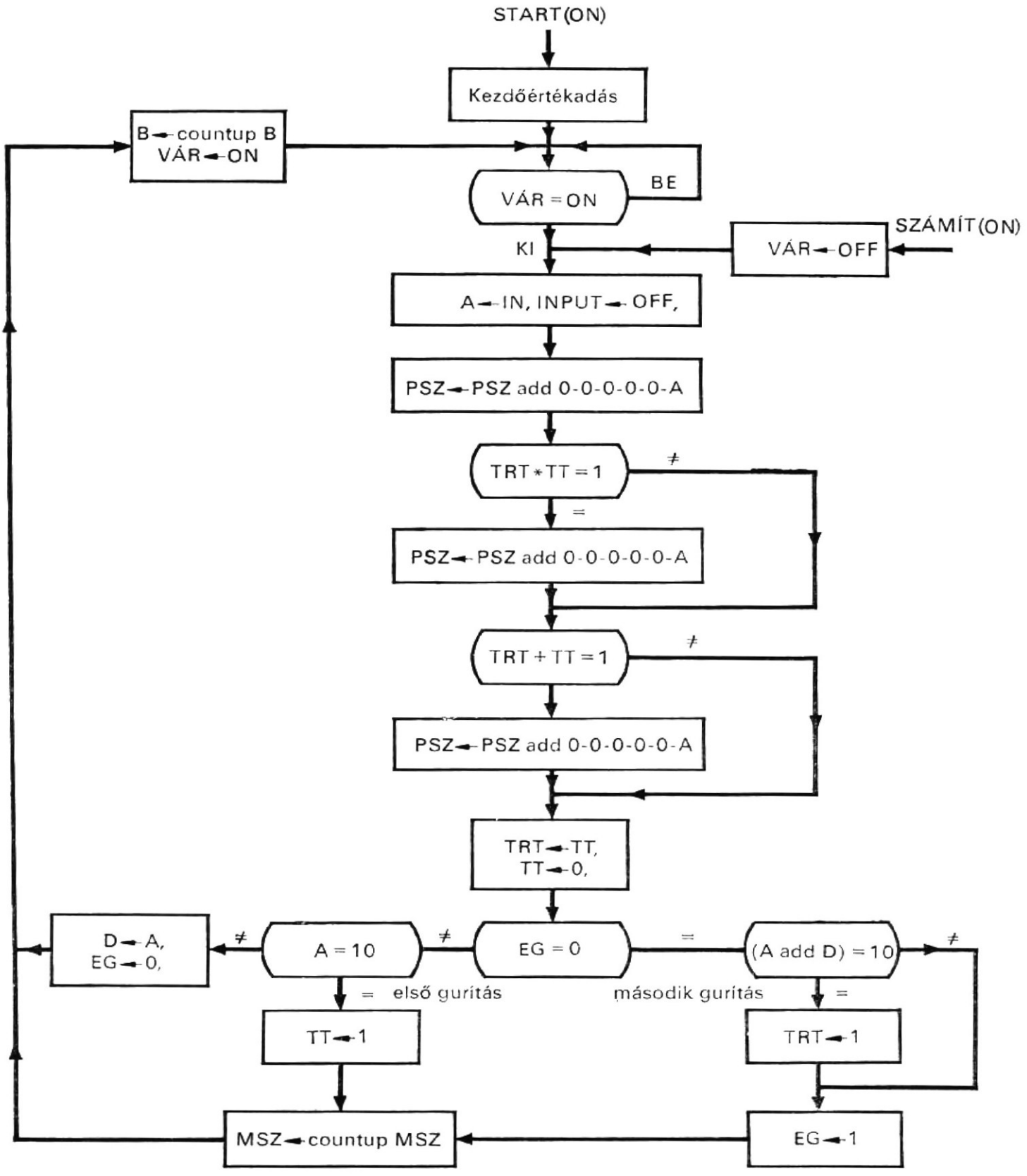
Comment, a tekeponteredmény-meghatározó számítógép struktúrája (2.30)

Register,	B(0—4),	$gurításszám-számláló
	A(0—3),	$bemenetregiszter
	MSZ(0—3),	$menetszámszámláló
	PSZ(0—8),	$pontszámregiszter az összegzett pontszámhoz
	D(0—3),	$első gurítást számláló regiszter
	T(0—2),	$vezérlésregiszter
	TT,	$telitalálatindikátor (ha 1)
	TRT,	$tartalékindikátor (ha 1)
	EG,	$első gurításindikátor (ha 1)
	M10,	$tizedik menet indikátor (ha 1)
Terminal,	IN(0—3),	$bemenetek, ha a gurításpontszám megjelenik
Decoder,	K(0—12)=T,	
Switch,	START(ON),	$kézi parancs az indításra
	SZÁMÍT(ON),	$kézi parancs a számításra
Light,	VÁR(ON, OFF),	$ha ON, gurításpontszámra való várakozást jelez
	INPUT(ON, OFF),	$ha ON, gurításpontszám megérkezését jelzi
	VÉGE(ON, OFF),	$ha ON, a játszma végét jelzi
Clock,	P	

2.8.4. Folyamatábrák

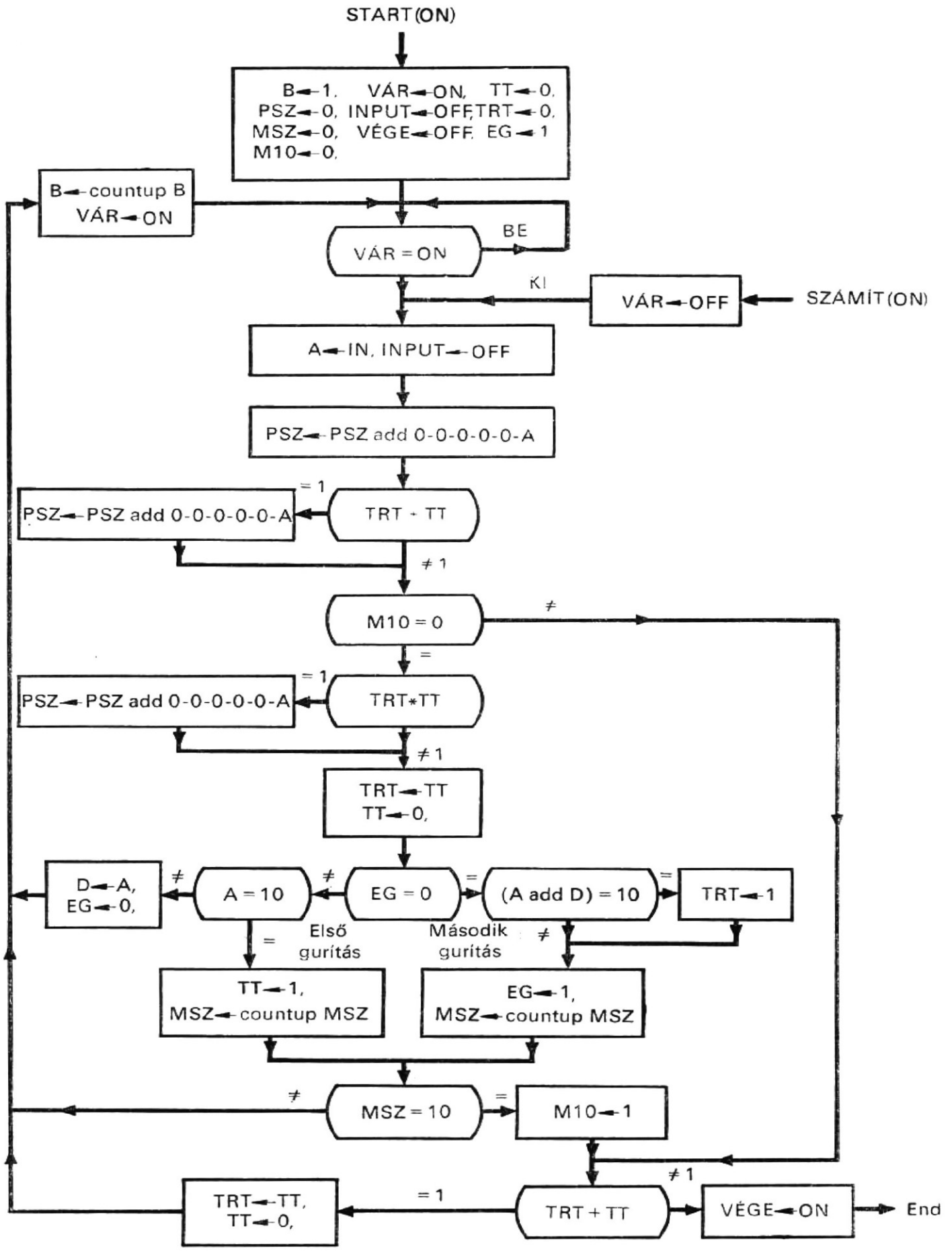
A tekejáték pontszámát meghatározó számítógép sorrendi műveleteit a 2.22. és 2.23 folyamatábrák mutatják. A 2.22. folyamatábra nem tartalmazza a végértékszámítást, a jobb áttekintőség miatt; a 2.23. ábra már ezt is tartalmazza.

Amint a 2.22. ábrából látható, amikor a START kapcsolót először kapcsolják be az ON állásba, a számítógép működésének megindítására a VÁR jelzőfény az ON állapotba kerül. A számítógép most várja a gurításpontszám beérkezését. Amikor a gurításpontszám beérkezik a bemeneti kapcsokra, az INPUT kijelző ON állapotba kerül. Ennek eredményeképpen a számítógép operátora bekapcsolja ON állásba a SZÁMÍT kapcsolót, ami viszont OFF állapotba helyezi a VÁR jelzőlámpát és ily módon megkezdődik a beérkező gurításpontszám összegezett ponteredményének számítása. A bemenő adatokat először az A regiszter fogadja, és az INPUT kijelző OFF állásba kerül. Ha akár a TT, akár a TRT regiszter 1-et tartalmaz, a gurítás pontszámát harmadszor is hozzáadják. Ekkor a TT regiszter 0 értéket kap, a TRT regiszter azonban csak akkor



2.22. ábra
Tekeeredmény-számítás folyamatábrája (a számításnak nincs befejezése)

lesz 1, ha az utolsó menetben telitalálat van. Ezután megvizsgálják az EG regisztert, annak meghatározására, vajon az aktuális gurításpontszám az első vagy a második gurítás pontszáma-e. Ha az első gurításé, akkor megvizsgálják, volt-e telitalálat. Ha volt telitalálat, a TT regiszter 1 értéket kap és az MSZ menetszámszámlálót 1-gyel megnövelik; egyébként pedig az A regiszterben levő első gurításpontszámot ideiglenesen a D regiszter tárolja és az EG regiszter 0 értéket kap, hogy jelezze, hogy a következő



2.23. ábra
Tekeeredmény-számítás folyamatábrája

gurítás második gurítás. Ha a pontszámot a második gurítás adja, akkor meghatározzák, volt-e tartalék: megvizsgálják, hogy az A és a D regiszterek tartalmának összege egyenlő-e 10-zel. Ha az összeg 10, a TRT regiszter 1 értéket kap; ezt követően az EG regisztert 1-re állítják és 1-gyel megnövelik az MSZ menetszámszámlálót. Ezzel befejeződött az aktuális gurításpontszám számítása. A következőkben 1-gyel megnövelik a B gurításszámszámlálót és a VÁR kijelző ON állapotba kerül, annak jelzésére, hogy a gép várja a következő gurításpontszámot.

Ha végelszámolást is beiktatunk, a 2.22. folyamatábra a 2.23. ábra szerint módosul. A végelszámoláshoz szükség van az utolsó menet felismerésére, és azután annak eldöntésére, hogy egy vagy két extra gurítás pontszáma kell-e. Ha ezek közül valamelyikre is szükség van, kiszámítják az extra gurítás vagy gurítások összegezett ponteredményét. Ezalatt a menetszámszámláló nem növekszik. Mint említettük, a gurításpontszámok egyszeri hozzáadása szükséges mindegyik extra golyónál, kivéve a telitalálat—telitalálat első gurításának esetét. Ebben az esetben két összeadásra van szükség. Amikor megkaptuk a végső ponteredményt, a VÉGE indikátor az ON állapotba kerül annak jelzésére, hogy a játszmának vége és a számítás befejeződött.

2.8.5. Az utasítások sorrendjének leírása

A 2.23. ábrán látható számítási algoritmust most a következő utasításokkal írjuk le:

```
Comment, itt kezdődik az összegzett ponteredmény számítása                (2.31)
/START(ON)/  B←1, PSZ←0, MSZ←0, M10←0, TT←0, TRT←0, EG←1,
               VÁR←ON, INPUT←OFF, VÉGE←OFF, T←0,
/SZÁMÍT(ON)/ VÁR←OFF, T←0,
/K(0)*P/     IF (VÁR=OFF) THEN (A←IN, INPUT←OFF, T←1),
/K(1)*P/     PSZ←PSZ add 0—0—0—0—0—A, T←2,
/K(2)*P/     IF (TRT+TT) THEN (PSZ←PSZ add 0—0—0—0—0—A), T←3,
/K(3)*P/     IF (M10=0) THEN (T←4) ELSE (T←10),
/K(4)*P/     IF (TRT*TT) THEN (PSZ←PSZ add 0—0—0—0—0—A), T←5,
/K(5)*P/     TRT←TT, TT←0, T←6,
/K(6)*P/     IF (EG=0) THEN (T←7) ELSE (T←8),
/K(7)*P/     IF ((A add D)=10) THEN (TRT←1), EG←1, MSZ←countup MSZ,
               T←9,
/K(8)*P/     IF (A=10) THEN (TT←1, MSZ←countup MSZ, T←9)
               ELSE (D←A, EG←0, T←11),
/K(9)*P/     IF (MSZ=10) THEN (M10←1, T←10) ELSE (T←11),
/K(10)*P/    IF (TRT+TT=1) THEN (T←12) ELSE (TRT←TT, TT←0, T←11),
/K(11)*P/    B←countup B, VÁR←ON, T←0,
/K(12)*P/    VÉGE←ON,
             END
```

Feladatok[6]

2.1. Szimulálja a (2.7) és (2.8) utasításokkal leírt soros paritásgenerátort!
2.2. A (2.7) és (2.8) utasításokkal leírt paritásbit-generálás meggyorsítható, ha az A(5,6) biteket és a PB regisztert egyszerre vizsgáljuk meg és ha az A regisztert kétbites pozíciókban lépjük jobbra. Határozza meg erre az esetre a folyamatábrát és írja le CDL utasításokkal az új paritásbit-generátort!
2.3. Szimulálja a (2.9) és (2.10) utasításokkal leírt soros komparátort!
2.4. Szimulálja a (2.9) és (2.11) utasításokkal leírt soros komparátort!
2.5. Két, előjeles 2-es komplemens ábrázolású számot hasonlítunk össze! Rajzoljon folyamatábrát, és írja le CDL-ben az algoritmust annak meghatározására, hogy az egyik szám nagyobb-e vagy kisebb, mint a másik, vagy egyenlő vele!
2.6. Szimulálja azt az eljárást, amely *n* egész szám között megkeresi a legnagyobb számot, amint azt a (2.12) utasítások leírják!
2.7. Ismételje meg a 2.6. feladatot a (2.13) leírására!
2.8. Szimulálja a (2.16) és (2.17) utasításokkal leírt prímszámgenerátort!
2.9. Mik az előnyei és hátrányai annak, ha osztás van a prímszámgenerátorban számlálás és kivonás helyett?
2.10. Szimulálja a (2.22) és (2.23) utasítások által leírt Gray—bináris kódkonvertert!
2.11. Szimulálja a (2.24), (2.25) és (2.26) utasításokkal leírt bináris—decimális konvertert!
2.12. Ha elhagyjuk a 2.16. ábrában a bináris—decimális konverterben a Q regisztert, milyen változásokat kell tenni a 2.18. folyamatábrában, feltételezve, hogy a 10-bites bináris bemenetet kezdetben a C—B—A regisztersor tárolja?
2.13. Adott egy 3-számjegyes BCD egész szám, írja le azt a konvertert, amely a BCD egész számot bináris egész számmá alakítja át!
a) Határozza meg a logikai hatásvázlatot!
b) Rajzolja meg a folyamatábrát!
c) Írja le a konverter működését CDL utasításokkal!
2.14. Szimulálja a (2.27) utasítások által leírt, tárolt átvitelű összeadást!
2.15. Szimulálja a (2.28) utasításokkal leírt nullára történő ellenőrzést!
2.16. Szimulálja a (2.29) utasításokkal leírt összehasonlítási tesztet!
2.17. Szimulálja a (2.30) és (2.31) utasításokkal leírt tekeponteredményt meghatározó számítógépet!

Irodalom

[1] BURKS, A.W.—GOLDSTINE, H. H. — von NEUMANN, J.: *Preliminary Discussion on the Design of an Electronic Computing Instrument.* Institute for Advanced Study, June, 1946.

[2] MERCER, R. J.: *Micro-Programming.* J. of the ACM, April, 1957, pp. 157...171.

[3] COULEUR, J. F.: *BIDEC-A Binary-to-Decimal or Decimal-to-Binary Converter.* IRE Transaction on Electronic Computers, December, 1958, pp. 313...316.

[4] CROY, J. E.: *Rapid Technique of Manual or Machine Binary-to-Decimal Integer Conversion Using Decimal Radix Arithmetic.* IRE Transaction on Electronic Computers, December, 1961, pp. 777.

[5] WILSON, M. C.: *Gray to Binary Converter.* Instruments & Control Systems, June, 1962, pp. 149... 150.

[6] ROZIER, C. P.: *Decimal-to-Binary Conversion Using Octal Radix Arithmetic.* IRE Transaction on Electronic Computer, October, 1962, pp. 708...709.

[6] Azoknál a feladatoknál, ahol szimulációt kérünk, valamilyen algoritmikus nyelv — pl. Fortran, Algol vagy PL/1 — alkalmazására utalunk.

[7] CHU, Y.: *Digital Computer Design Fundamentals.* New York: McGraw-Hill Book Company, 1962, pp. 13...14; 143...145.

[8] CHU, Y.: *An Algol-like Computer Design Language.* Comm. of ACM, October, 1965, pp. 607...615.

[9] CHU, Y.—MESZTENYI, C. K.: *Macro Logic Design of Digital Computers.* Technical Report 67—57, Computer Science Center, University of Maryland, November, 1967.

[10] KNUTH, D. E.: *The Art of Computer Programming.* Vol. 1. Reading, Massachusetts: Addison-Wesley Publishing Co., Inc., 1968, pp. 143...145.

[11] VAN TASSEL, D.: *The Chinese Remainder Theorem.* Computer Decisions, November, 1969, pp. 69.

[12] SINGLETON, R. C.: *A Prime Number Generator using the Treesort Principle, Algorithm 356.* Communications of the ACM, October, 1969, pp. 563.

[13] SINGLETON, R. C.: *An Efficient Prime Number Generator, Algorithm 357.* Communications of the ACM, October, 1969, pp. 563.

3.
Mikroprogramozás

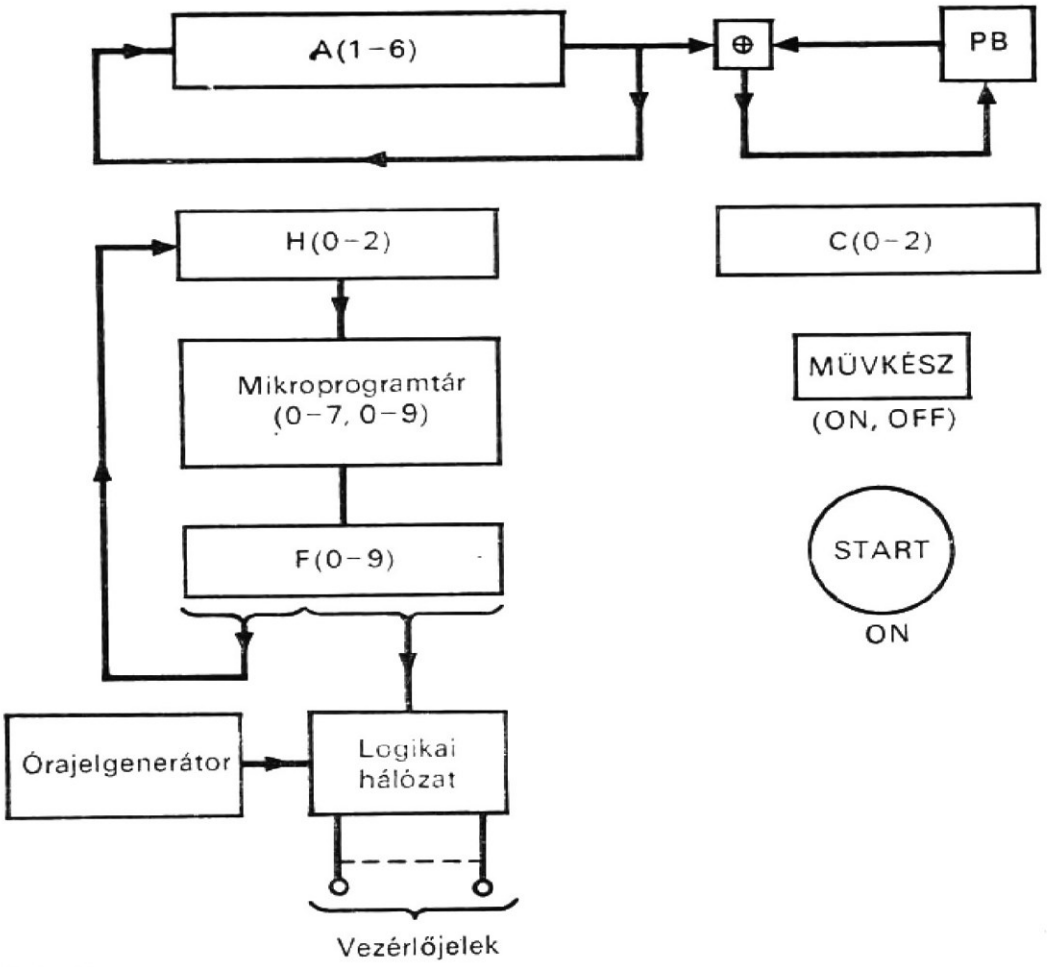
A vezérlésnek azt a típusát, amely a 2. fejezetben leírt feladatok mikroműveleteinek előírt sorrendjét biztosítja, *sorrendi (szekvenciális) vagy huzalozott vezérlésnek* nevezik; azért, mert a vezérlőjelek sorrendjeit huzalozott logikai hálózat állítja elő. Az angol M.V. Wilkes [1] 1951-ben egy másik típusú vezérlést javasolt, amit *mikroprogramozott vezérlésnek* neveznek [2, 5, 8]. A mikroprogramozott vezérlésnek az az elve, hogy a huzalozott logikai hálózattal végzett vezérlést programmal helyettesíti, és a programot olyan tárban tárolja, amely csak kiolvasásra szolgál. Ezt a tárat mikroprogramtárnak vagy a *vezérlőtárnak* nevezik. Azt a programot, amely a mikroprogramtárban helyezkedik el, *mikroprogramnak* nevezik. A mikroprogram készítése a *mikroprogramozás.* Ha a digitális számítógép mikroprogram-vezérlést alkalmaz, akkor az *mikroprogramozott számítógép.*

Ez a fejezet bevezetést ad a mikroprogramozott vezérlés és a mikroprogramozás alapfogalmaiba és elveibe, néhány példa bemutatásával. Az első példa segítségével megismerkedünk a mikroprogram-vezérléssel és ennek kapcsán megvizsgáljuk a sorrendi huzalozott logikai vezérlés és a mikroprogram-vezérlés közötti különbséget. A második példa egy mikroprogramozott számítógépet ír le, míg a harmadik egy tárolt programú logikai számítógépet. Az utolsó példa a főtár, a mikroprogramtár és az órajelgenerátor ciklusai közötti időzítési problémákkal foglalkozik.

3.1.
Paritásgenerátor

A soros üzemű paritásgenerátort a 2. fejezetben már tárgyaltuk. A generátor logikai hatásvázlata a 2.1. ábrán, működése a 2.2. folyamatábrán látható, míg teljes leírását a (2.7) és (2.8) utasításokban adtuk meg. A 2. fejezetben vázolt generátor sorrendi vagy huzalozott logikai vezérlést használ, míg az alábbiakban leírásra kerülő generátorhoz mikroprogram-vezérlést alkalmazunk.

A (2.8) utasítássorozat számára a T fázisregiszter állítja elő a vezérlés kívánt sorrendjét. A T regiszter minden bitje egy vagy két mikroművelet vezérlését végzi. A T(1) bit a KIZÁRÓ—VAGY és a számlálás mikroműveleteit vezérli. A T(2) bit a ciklikus léptetés balra mikroműveletre ad parancsot. A T(3) bit irányítja a feltételes elága-



3.1. ábra
Soros paritást mikroprogramozott vezérléssel előállító hálózat hatásvázlata

zás mikroműveletet, míg a T(4) bit a MŰVKÉSZ indikátort kapcsolja be. A következő öt mikroműveletre:

PB←A(6)⊕PB, (3.1)
C←countup C,
A←cir A,
IF (C=6) THEN (...) ELSE (...),
MŰVKÉSZ←ON

a mikroprogram-vezérlés esetén is szükség lesz, mivel ezek végzik a paritásbit generálásának funkcióját.

3.1.1. Mikroprogramozott vezérlés logikai hatásvázlata

A 3.1. ábrán mikroprogrammal vezérelt, soros paritásgenerátor logikai hatásvázlatát láthatjuk. Ez a struktúra a következő utasításokkal írható le:

Comment, paritásgenerátor mikroprogramozott vezérléssel (3.2)

Register,	A(1–6),	$léptetőregiszter
	C(0–2),	$számláló
	PB,	$paritásbit-regiszter

	H(0—2),	$címregiszter
	F(0—9),	$mikroutasítás-regiszter
Memory,	MT(H)=VT(0—7,0—9)	$mikroprogramtár
Switch,	START(ON),	$start-kapcsoló
Light,	MŰVKÉSZ(ON, OFF),	$befejezést jelző tároló
Clock,	P(1—2),	$kétfázisú órajelgenerátor

A (3.2) utasításokkal leírt hálózatban a T bázisregisztert az MT mikroprogramtár váltja fel, a H címregiszterrel és az F mikroutasítás-regiszterrel együtt, továbbá kétfázisú órajelgenerátort használunk az egyfázisú helyett.

3.1.2. Mikroutasítás formátuma

A (3.1)-ben szereplő öt mikroműveleten kívül további három mikroművelet:

```
F←MT(H),                                                    (3.3)
H←countup H,
IF (C=6) THEN (H←countup H) ELSE (H←F(0—2))
```

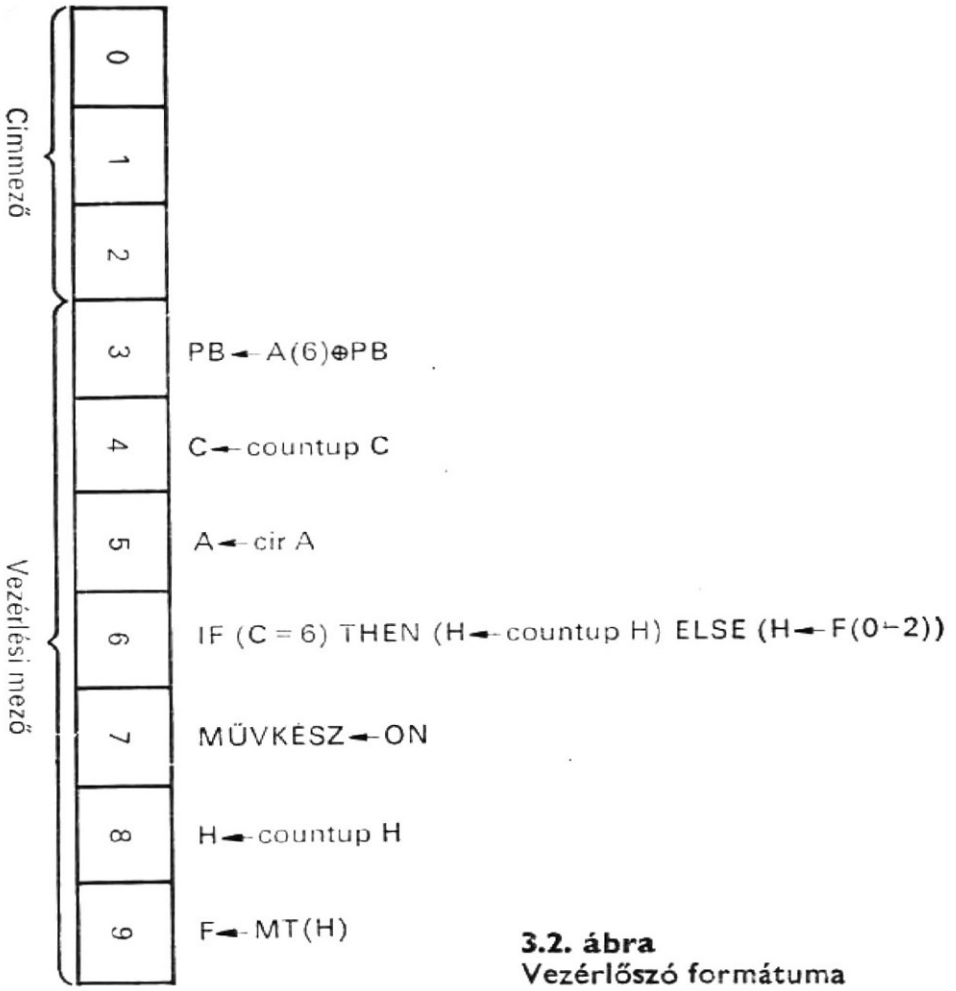
szükséges a mikroprogramtár alkalmazása miatt. Az első mikroművelet kiolvas egy szót a mikroprogramtárból, és beírja azt az F mikroutasítás-regiszterbe. A második mikroművelet 1-gyel megnöveli a H címregiszter tartalmát. A harmadik egy feltételes mikroművelet, amely ellenőrzi és elágaztatja a sorrendet; ez a mikroművelet a (3.1) utasításban levőt helyettesítheti. Így összesen hét mikroműveletünk van.

A mikroprogramtárban levő egyes szavakat *vezérlőszavaknak* vagy *mikroutasításoknak* (instrukciók) nevezik. A mikroprogram a mikroutasítások (vezérlőszavak) sorozata a mikroprogramtárban. A mikroutasítás formátuma a 3.2. ábrán látható. Ez 10 bitet tartalmaz, az F(0—2) a mikroutasítás címmezeje. A maradék hét bit közül mindegyik a hét mikroművelet valamelyikéhez van hozzárendelve. Ha az F(j) bit 1 (ahol $3 \leqq j \leqq 9$), a j-edik mikroművelet parancsjele generálódik. Minthogy egynél több F(j) bit lehet egyenlő 1-gyel, egynél több mikroművelet jöhet szimultán létre. Példaképpen négy vezérlőszót mutatunk be a 3.3. ábrán, ahol a vesszők csupán az olvashatóságot javítják, de ténylegesen a tárban nem léteznek. Ez a négy vezérlőszó — amint a továbbiakban láthatjuk — alkotja a paritásgeneráló hálózat mikroprogramját.

3.1.3. A vezérlés sorrendjének meghatározása

A paritásgenerátor sorrendi működését a 2.2. folyamatábrán már meghatároztuk. A sorrendet előíró vezérlőjelek a következőképpen kaphatók. A folyamatábrában minden mikroműveletet — a mikroutasítás-formátumnak megfelelően — hozzárendelünk valamelyik vezérlőbithez. Ezek a mikroműveletek a P(1) órajelfázisban aktiválódnak, kivéve azt a mikroműveletet, amely a mikroutasítást hívja le a mikroprogramtárból. Ez a P(2) órajelfázis alatt történik. A vezérlőbiteknek és az órajelfázisoknak a megválasztása után egy-egy vezérlőjelet lehet rendelni az egyes mikroműveletekhez. A hét mikroműveletet a következő utasításokkal lehet leírni:

```
/F(3)*P(1)/    PB←A(6)⊕PB,                                  (3.4)
/F(4)*P(1)/    C←countup C,
```



3.2. ábra
Vezérlőszó formátuma

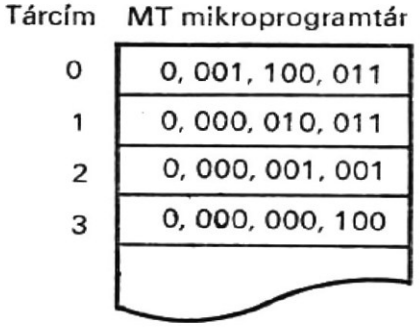
/F(5)∗P(1)/ A←cir A,
/F(6)∗P(1)/ IF (C=6) THEN (H←countup H) ELSE (H←F(0—2)),
/F(7)∗P(1)/ MŰVKÉSZ←ON,
/F(8)∗P(1)/ H←countup H,
/F(9)∗P(2)/ F←MT(H),

A fenti utasításokkal a 2.2. ábrán levő vezérlési sorrendet most a következőképpen lehet leírni:

/START(ON)/ PB←1, H←0, C←0, MŰVKÉSZ←OFF, F ←1, (3.5)
/F(9)∗P(2)/ F←MT(H),

Comment, az aktuális paritásbit generálása

/F(3)∗P(1)/ PB←A(6)⊕PB,
/F(4)∗P(1)/ C←countup C,
/F(8)∗P(1)/ H←countup H,
/F(9)∗P(2)/ F←MT(H),



3.3. ábra
Mikroprogram a mikroprogramtárban

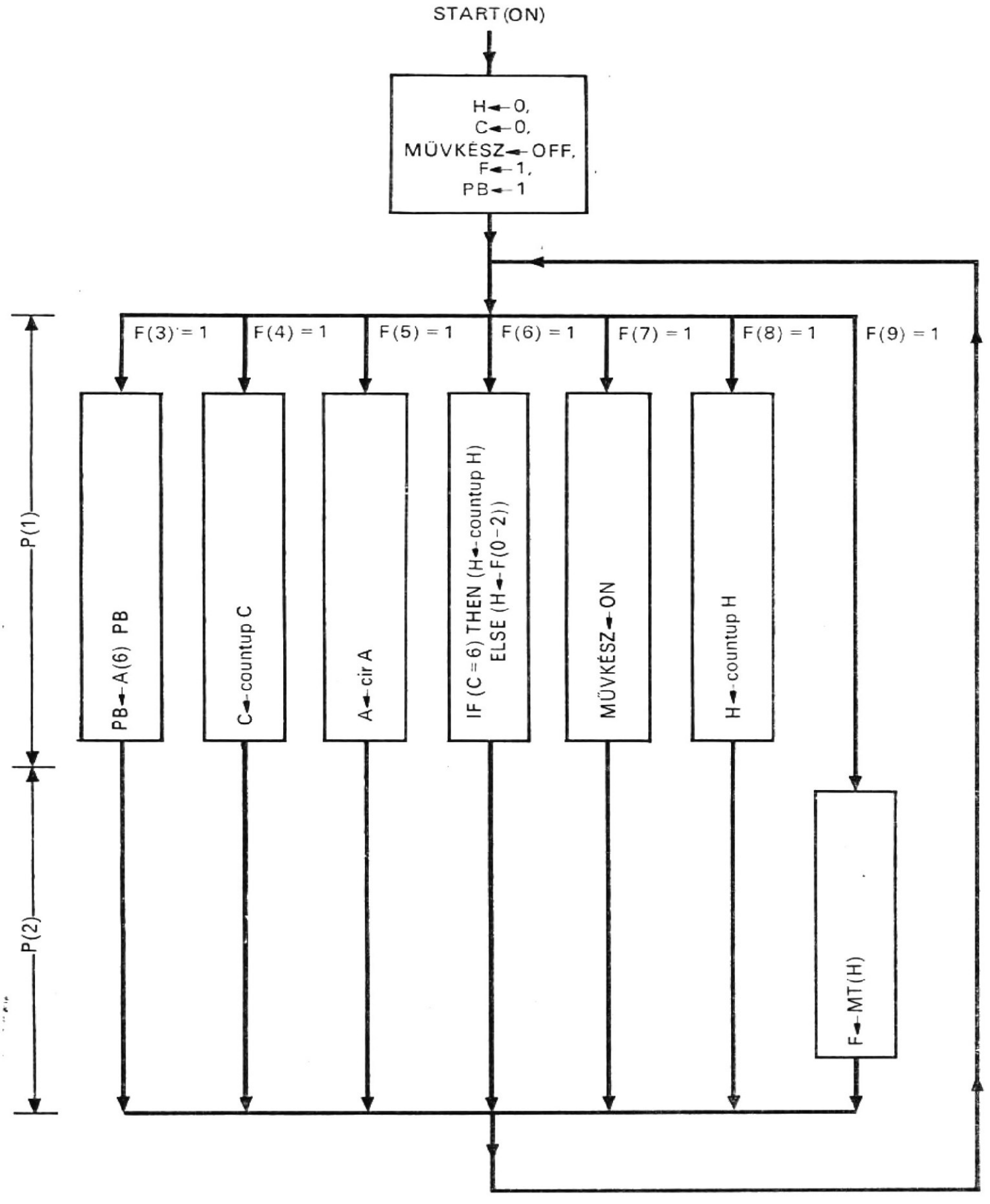
```
Comment, ciklikus léptetés jobbra
/F(5)*P(1)/    A←cir A,
/F(8)*P(1)/    H←countup H,
/F(9)*P(2)/    F←MT(H),
Comment, teszt és elágazás
/F(6)*P(1)/    IF (C=6) THEN (H←countup H) ELSE (H←F(0—2)),
/F(9)*P(2)/    F←MT(H),
Comment, befejezés kijelzése
/F(7)*P(1)/    MŰVKÉSZ←ON,
               END
```

3.1.4. Mikroprogram

A (3.5)-ben az utasítások öt csoportra vannak osztva. Mindegyik csoport élén egy comment (megjegyzés) van, és az egyes csoportok a 2.2. folyamatábra öt blokkja közül az egyikben levő mikroműveletet (műveleteket) hajtják végre. Az első comment utasítás arra a mikroművelet-csoportra utal, amelyik a műveletsorozat végrehajtását kezdeményezi, ha a START kapcsolót az ON állásba fordítjuk (kezdetiérték-beállítás). Ezek hatására a PB regiszterbe 1 kerül, 0-át írunk a H és C regiszterbe a MŰVKÉSZ indikátor OFF állapota jut és az F regiszterbe 0001_8 érték íródik. Ez utóbbi egy mikroutasításnak a mikroprogramtárból való kiolvasására vonatkozó mikroművelet parancskódja. A második comment utáni utasítás négy mikroműveletből álló csoportra utal, amelyet a 3.3. ábrán látható mikroprogramtárban levő első mikroutasítás (0143_8) határoz meg. Ez a mikroutasítás négy 1-et tartalmaz, (l. még 3.2. ábrát is), ezek vezérlik a következő négy mikroműveletet: paritásbit előállítása, a C inkrementálása, a H inkrementálása, valamint kiolvasás a mikroprogramtárból. A harmadik comment utáni utasítás arra a mikroműveletcsoportra vonatkozik, amelyet a mikroprogramtárban levő második mikroutasítás (0023_8) specifikál. Ez a vezérlőszó három 1-et tartalmaz, amelyek a következő három mikroműveletet vezérlik: ciklikus léptetés balra, a H inkrementálása, és kiolvasás a mikroprogramtárból. A negyedik comment utáni utasítás arra a mikroműveletcsoportra utal, amelyet a mikroprogramtárban levő harmadik mikroutasítás (0011_8) specifikál. Ez a mikroutasítás két 1-et tartalmaz, amelyek a következő két mikroműveletet vezérlik: teszt—elágazás és kiolvasás a mikroprogramtárból. Az utolsó comment utáni utasítás arra a mikroműveletre utal, amely a MŰVKÉSZ kijelzőt ON állásba viszi, amint azt a mikroprogramtárban levő negyedik vezérlőszó (0004_8) előírja. A (3.5)-ben levő utasítások tehát leírják a vezérlést a 3.3. ábrán bemutatott mikroprogram mikroutasításaival.

3.1.5. Vezérlési ciklus

Amikor a mikroprogram végrehajtásra kerül, a soros paritásgenerátor működése a 2.2. folyamatábrát követi. A mikroprogram egyes mikroutasításainak végrehajtását a 3.4. ábrán látható, ciklikus vezérlési ábrával lehet leírni. Az ebben szereplő vezérlő huroknak annyi ága van, amennyi a mikroműveletek száma. A hurok egyes ciklusaiban



3.4. ábra.
Soros paritásbit-generátor vezérlésének állapotdiagramja

a mikroutasítás által specifikált egy vagy több mikroművelet kerül szimultán végrehajtásra, az első órajelfázis alatt. A következő mikroutasítás kiolvasása a mikroprogramtárból a második órajelfázis alatt megy végbe. A következő mikroutasítás helye úgy határozható meg, hogy vagy a H regiszter értékét megnövelik, vagy pedig az F(0—2) címmezőt átviszik a H regiszterbe. A mikroprogrammal vezérelt számítógépek sebességét jelenleg ennek a vezérlőhuroknak a végrehajtási sebessége korlátozza.

3.1.6. Összehasonlítás

A 2.1. ábrán levő hatásvázlatban a vezérlést a T regiszterrel vezérelt logikai hálózat valósítja meg; ennek működési sebességét a kombinációs logikai hálózat sebessége korlátozza. A 3.1. ábra szerinti realizálásban a mikroprogramtárban elhelyezkedő mikroprogram végzi a vezérlést; ennek működési sebességét a mikroprogramtár sebessége korlátozza. Minthogy a technika jelenlegi állása szerint a mikroprogramtár lényegesen lassabban működik, mint a logikai áramkör, a mikroprogram-vezérlés nem tudja a leggyorsabb működési sebességét nyújtani. Másrészt, a sorrendi logikai vezérlés költségei valamelyest arányosak a vezérlés bonyolultságával, míg a mikroprogramozott vezérlés árát lényegében a mikroprogramtár ára határozza meg, és — korlátok között — viszonylag érzéketlen a vezérlés bonyolultsága iránt.

Így jelen pillanatban azt a következtetést vonhatjuk le, hogy a mikroprogramozott vezérlés esetleg nem előnyös nagy sebességek esetén, viszont rugalmasságot és megtakarítást kínál bonyolultabb vezérlések realizálásakor.

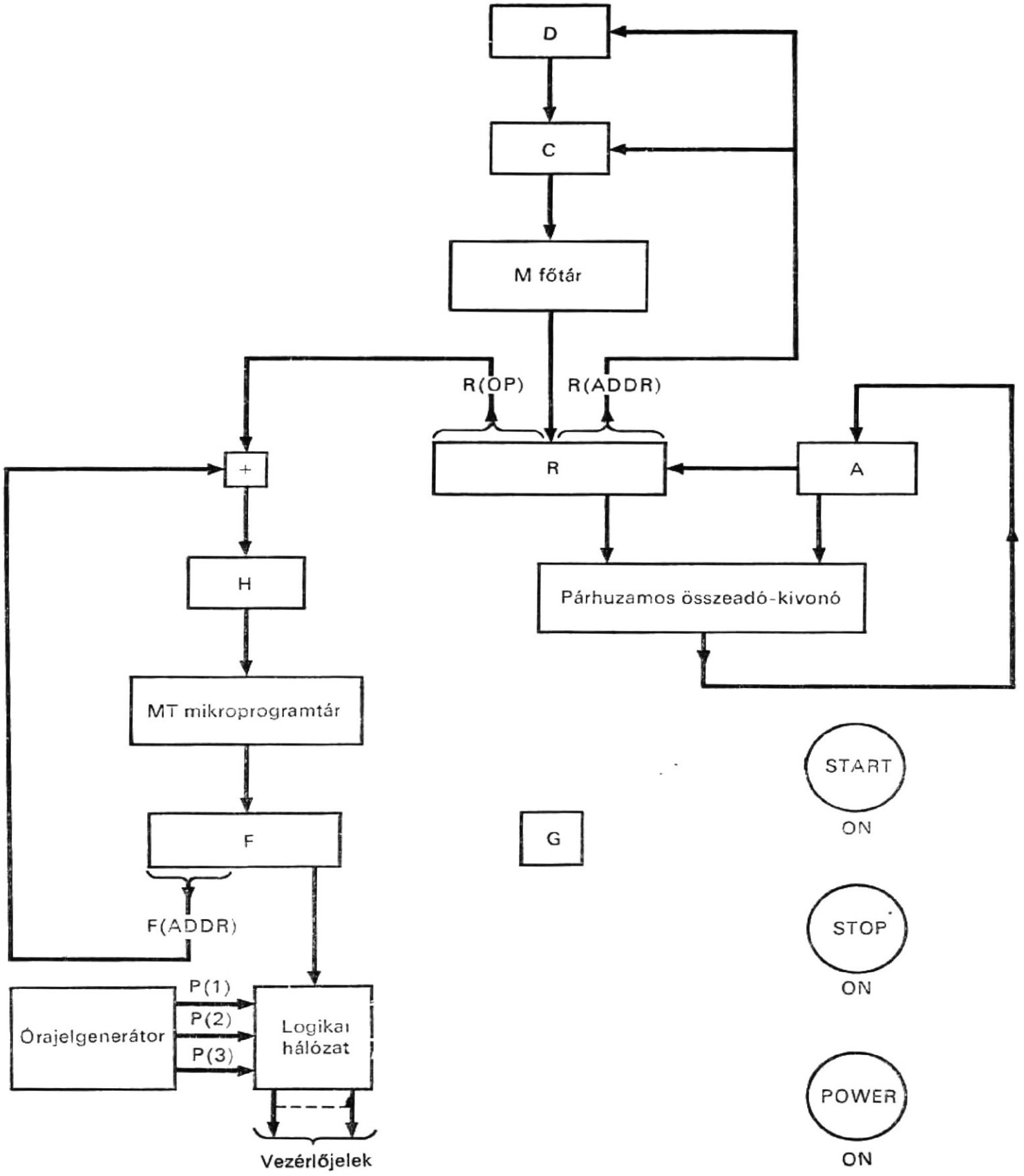
3.2. Mikroprogramozott számítógép

A mikroprogramozott vezérlés elvének bemutatása után most egy egyszerű, mikroprogramozott vezérlésű számítógépet mutatunk be. Ez legyen ugyanaz a tárolt programú számítógép, amellyel az 1. fejezetben már megismerkedtünk, azzal az eltéréssel, hogy vezérlése mikroprogramozott a huzalozott (sorrendi logikai) vezérlés helyett.

3.2.1. Logikai hatásvázlat

A számítógép logikai hatásvázlata a 3.5. ábrán látható. A számítógépnek van egy M véletlen hozzáférésű tára, a C címregiszterrel és R tartalomregiszterrel. Az M főtár kapacitása 32 768, egyenként 24-bites szó. A számítógéphez tartozik még a D utasításszámláló, az A akkumulátorregiszter, a G start—stop vezérlés regisztere, valamint a START, STOP és POWER kapcsolók és a P háromfázisú órajelgenerátor. A mikroprogram az MT mikroprogramtárban van, amelynek H a címregisztere és F a tartalomregisztere. Ezt a struktúrát írjuk most le a következő deklarációs utasításokkal:

```
Comment, mikroprogramozott vezérlésű számítógép                    (3.6)
Register,    R(0—23),        $főtár tartalomregisztere
             A(0—23),        $akkumulátorregiszter
```



3.5. ábra
Egyszerű, mikroprogramozott számítógép logikai hatásvázlata

	C(0—14),	$az M főtár címregisztere
	D(0—14),	$utasításszámláló
	F(0—23),	$mikroprogramtár tartalomregisztere
	H(0—9),	$az MT mikroprogramtár címregisztere
	G,	$start—stop vezérlés regisztere
Subregister,	R(OP)=R(0—5),	$az R regiszter műveletikód-része

```
                R(ADDR)=R(9—23),  $az R regiszter címrésze
                F(ADDR)=F(0—9),   $az F regiszter címrésze
Memory,         M(C)=M(0—32767,0—23)  $főtár
                MT(H)=MT(0—1023,0—23)  $mikroprogramtár
Switch,         POWER(ON),        $tápfeszültség-kapcsoló,
                START(ON),        $start-kapcsoló
                STOP(ON),         $stop-kapcsoló
Clock,          P(1—3),           $háromfázisú-óra
```

3.2.2. Vezérlőjelek

A főtár ciklusát úgy választjuk meg, hogy egybeessen az órajelciklussal. A főtár egy teljes ciklusában három lépés van, amelyek mindegyikét egy órajelfázis vezérel. Amikor egy szót kell kiolvasni a főtárból, mind a tárcímnek a címregiszterbe való beírása, mind pedig a tárkiolvasási működés kezdeményezése a megelőző órajelciklus P(3) fázisában történik. A szó az R regiszterben az aktuális órajelciklus P(1) órajelfázisában áll rendelkezésre. Így a következőket írhatjuk le:

```
/P(3)/    C←R(ADDR) vagy C←D,  $a főtárciklus kezdete          (3.7)
/P(1)/    R←M(C),
/P(2)/                         $a főtárciklus vége
```

Amikor egy szót be kell írni a főtárba, mind a tárcím átvitele a címregiszterbe, mind pedig a tárbeírás műveletének kezdeményezése a megelőző órajelciklus P(3) fázisában történik. A szó beírása a tár az R regiszterből az aktuális órajelciklus P(1) fázisában megy végbe. Így a következőket írhatjuk fel:

```
/P(3)/    C←R(ADDR) vagy C←D,  $a főtárciklus kezdete
/P(1)/                                                       (3.8)
/P(2)/    M(C)←R               $a főtárciklus vége
```

A mikroprogramtár csak olvasásra használt tároló. Ciklusát úgy választják meg, hogy az órajelciklussal a következő módon essék egybe. A tárcímnek a címregiszterbe való átvitele és a tárból való kiolvasási művelet kezdeményezése egyaránt a P(2) órajelfázisban megy végbe, míg a szó az F regiszterben a P(3) órajelfázisban áll rendelkezésre. Így a következőket kapjuk:

```
/P(1)/
/P(2)/    H←R(OP) vagy H←F(ADDR),  $a mikroprogramtár-ciklus kezdete
                                                             (3.9)
/P(3)/    F←MT(H)                  $a mikroprogramtár-ciklus vége
```

A (3.9)-ben az időzítés olyan, hogy a mikroutasítás a P(3) órajelfázisban kerül kiolvasásra a mikroprogramtárból és így a végrehajtási sorrend (mint például egy ADD vagy SUB műveletsorozat, amint a következőkben látni fogjuk) a P(1) órajelfázisban el tud kezdődni.

3.2.3.
A mikroutasítás formátuma

A 3.1. táblázat a mikroutasítás formátumát mutatja. A mikroutasítás 24 bitből áll, három mezője van. Az F(0—9) mező a mikroutasítás címét tartalmazza elágazás céljából; az F(10—22) mezőben foglalnak helyet a vezérlőbitek; az F(23) bit nincs felhasználva. A vezérlőbitek a megfelelő órajelfázissal együtt hozzá vannak rendelve az 1.5. folyamatábrán szereplő mikroműveletekhez, valamint a mikroprogram vezérléshez szükséges további mikroműveletekhez.

3.1. táblázat

Vezérlőszó formátuma

Vezérlőbit	Órajelfázis	Mikroműveletek
F(0—9)		mikroprogramtárcím
F(10)	P(1)	R←M(C)
F(11)	P(2)	H←R(OP), D←countup D
F(12)	P(3)	IF (G) THEN (F←MT(H), C←R(ADDR) ELSE (H←0, C←0, D←0, R←0)
F(13)	P(2)	H←F(ADDR)
	P(3)	C←D, F←MT(H)
F(14)	P(1)	R←A
	P(2)	M(C)←R
F(15)	P(2)	A←A add R
F(16)	P(2)	A←A sub R
F(17)	P(1)	D←R(ADDR)
F(18)	P(1)	IF (A(0)) THEN (D←R(ADDR))
F(19)	P(1)	A←Shr A
F(20)	P(1)	A←cil A
F(21)	P(1)	A←0
F(22)	P(1)	G←0
F(23)		nincs felhasználva

3.2.4.
A műveletek sorrendjének leírása

A 3.1. táblázatban megadott mikroutasítás-formátummal az 1.5. ábra számítógépének folyamatábrája a következő utasításokkal írható le (l. még az 1.2. táblázatot).

Comment, a mikroprogramozott vezérlésű számítógép működésének leírása (3.10)

Comment, kezdeti értékek adása

/POWER(ON)/ G←0, F←0, H←9, C←0, D←0, R←0,

Comment, start-művelet

/START(ON)/ G←1, F(12)←1, F(0—11,13—22)←0,

Comment, stop-művelet

/STOP(ON)/ G←0,

```
Comment, utasításleírás, amikor H=9.

/F(10)*P(1)/    R←M(C),
/F(11)*P(2)/    H←R(OP), D←countup D,
/F(12)*P(3)/    IF (G) THEN (F←MT(H), C←R(ADDR))
                       ELSE (H←9, C←0, D←0, R←0),

Comment, ADD műveletének összetevői, amikor H=0.

C/F(10)*P(1)/   R←M(C),
/F(15)*P(2)/    A←A add R,
/F(13)*P(2)/    H←F(ADDR),
/F(13)*P(3)/    C←D, F←MT(H),

Comment, SUB(kivonás) műveletének összetevői, amikor H=1

C/F(10)*P(1)/   R←M(C),
/F(16)*P(2)/    A←A sub R,
C/F(13)*P(2)/   H←F(ADDR),
C/F(13)*P(3)/   C←D, F←MT(H),

Comment, JOM műveleteinek összetevői, amikor H=2

/F(18)*P(1)/    IF (A(0)) THEN (D←R(ADDR)),
C/F(13)*P(2)/   H←F(ADDR),
C/F(13)*P(3)/   C←D, F←MT(H),

Comment, STO művelet összetevői, amikor H=3

/F(14)*P(1)/    R←A,
/F(14)*P(2)/    M(C)←R,
C/F(13)*P(2)/   H←F(ADDR),
C/F(13)*P(3)/   C←D, F←MT(H),

Comment, JMP művelet összetevői, amikor H=4

/F(17)*P(1)/    D←R(ADDR),
C/F(13)*P(2)/   H←F(ADDR),
C/F(13)*P(3)/   C←D, F←MT(H),

Comment, SHR művelet összetevői, amikor H=5

/F(19)*P(1)/    A←shr A,
C/F(13)*P(2)/   H←F(ADDR),
C/F(13)*P(3)/   C←D, F←MT(H),

Comment, CIL művelet összetevői, amikor H=6

/F(20)*P(1)/    A←cil A,
C/F(13)*P(2)/   H←F(ADDR),
C/F(13)*P(3)/   C←D, F←MT(H),

Comment, CLA művelet összetevői, amikor H=7

C/F(10)*P(1)/   R←M(C),
/F(21)*P(1)/    A←0,
C/F(15)*P(2)/   A←A add R,
C/F(13)*P(2)/   H←F(ADDR),
C/F(13)*P(3)/   C←D, F←MT(H),
```

Comment, STP művelet összetevői, amikor H=8

```
/F(22)*P(1)/     G←0,
C/F(13)*P(2)/    H←F(ADDR),
C/F(13)*P(3)/    C←D, F←MT(H),
                 END
```

A fenti start-műveletek magyarázatra szorulnak. Amikor a START kapcsolót ON állásba fordítják, mind a G regiszter, mind az F(12) bit 1 értéket kap. Ez a következő mikroművelet végrehajtását eredményezi:

```
IF (G) THEN (F←MT(H), C←R(ADDR)                    (3.11)
ELSE (H←0, C←0, D←0, R←0)
```

A START kapcsoló elfordítását követő órajelfázis P(1), P(2) vagy P(3) lehet. Ha P(1) vagy P(2), semmi sem történik. Ha P(3), a fenti feltételes mikroutasítás kerül végrehajtásra. Minthogy a G értéke 1 lett, a fenti mikroutasítás végrehajtásával kiolvasásra kerül a tárból az első mikroutasítás, és megkezdődik a számítógép működése. Másrészt, ha a STOP kapcsolót OFF állásba fordítják a számítógép működése közben, a számítógép addig folytatja az aktuális utasítás végrehajtását, amíg el nem éri az utasításlehívó ciklus P(3) órajelfázisát. Ekkor egyetlen mikroutasítás sem kerül kiolvasásra a mikroprogramtárból, és a számítógép várakozási állapotban van, egészen addig, amíg a START kapcsolót el nem fordítják.

A fenti végrehajtási utasítások közül néhánynak az első pozíciójában egy „C" betű jelenik meg. A C betű jelenléte az utasításban azt jelöli, hogy az „comment" utasítás. Ezekre a comment utasításokra nincs szükség, mivel valahol máshol már megjelentek, viszont a leírást sokkal olvashatóbbá teszik.

3.2.5. Mikroprogram

Amint láthattuk, a (3.10) sorrendi leírás 13 csoportra oszlik. Minden csoport élén egy comment utasítás áll. A csoportok egy-egy speciális műveletsorozat mikroműveleteit adják meg. Az első három csoport az indítási és leállítási műveletek kézi vezérlését írja le, míg a negyedik csoport az utasításlehívási sorozatot specifikálja. A maradék csoportok közül mindegyik egy-egy gépi utasítás végrehajtását írja le a kilenc közül. Az első három csoportban levő mikroműveleteket kézzel vezérlik és így a mikroprogramban nem jelennek meg. A többi csoportban a mikroműveleteket egy-egy mikroutasítás vezérli. A 10 csoport számára 10 mikroutasítás van a mikroprogramban: egy az utasításlehívási művelet sorozathoz tartozik és a többi kilenc gépi utasításhoz. A számítógép mikroprogramját a 3.2. táblázat mutatja.

Minthogy a mikroprogram mérete kihat a mikroprogramtár kapacitására és sebességére, a mikroprogramozott vezérlést úgy szervezik, hogy a vezérlőszavaknak mind a számát, mind a hosszát minimalizálni lehessen. A (3.10) leírásban 20 mikroművelet van (nem számítva ide a kézi vezérlésűeket), amint az a 3.3. táblázatban is látható. Ha egy-egy vezérlőbitet rendelünk minden egyes mikroművelethez, a vezérlőszó további 20 bitet igényelne a 10 címbiten kívül, azaz hossza összesen 30 bit lenne. A 3.1. táblázatban levő mikroutasítás viszont csak 24-bites szóból áll (amellett 1 bit nincs is kihasználva). Ezt a 30-ról 24 bitre történő csökkenést több úton érhetjük el. Az egyik út az, hogy a mikroműveleteket egy vezérlőjel alá csoportosítjuk. Például, a 3.1. táblázatban két mikroműveletet csoportosítottunk az F(11)*P(2) vezérlőjel alá, mivel azok egyidőben fordulnak elő. Azonban ez a redukció az egyes mikroműveletek egyedi al-

3.2. táblázat

A mikroprogramozott számítógép mikroprogramja

H	Kód	F(0—9)	F(10)	F(11)	F(12)	F(13)	F(14)	F(15)	F(16)	F(17)	F(18)	F(19)	F(20)	F(21)	F(22)
9	FETCH	0	1	1	1	0	0	0	0	0	0	0	0	0	0
0	ADD	9	1	0	0	1	0	1	0	0	0	0	0	0	0
1	SUB	9	1	0	0	1	0	0	1	0	0	0	0	0	0
2	JOM	9	0	0	0	1	0	0	0	0	1	0	0	0	0
3	STO	9	0	0	0	1	1	0	0	0	0	0	0	0	0
4	JMP	9	0	0	0	1	0	0	0	1	0	0	0	0	0
5	SHR	9	0	0	0	1	0	0	0	0	0	1	0	0	0
6	CLS	9	0	0	0	1	0	0	0	0	0	0	1	0	0
7	CLA	9	1	0	0	1	0	1	0	0	0	0	0	1	0
8	STP	9	0	0	0	1	0	0	0	0	0	0	0	0	1

Megjegyzés:
H=mikroprogramtárcím.
F(0-9)=a következő cím a mikroprogramtárban, decimális számban megadva.
F(10-22)=vezérlőbitek.
F(23) nincs felhasználva és feltüntetve.

3.3. táblázat

A mikroprogramozott számítógép mikroműveletei

Mikroműveletek	Mikroutasítások				
	műveleti kód	vizsgálati feltétel	operandusok vagy konstansok		
			1-ső	2-ik	3-ik
R←M(C)	0		R	M	
M(C)←R	1		M	R	
F←MT(H)	2	G	F	MT	
H←R(OP)	3		H	R(OP)	
H←F(ADDR)	4		H	F(ADDR)	
C←R(ADDR)	5	G	C	R(ADDR)	
D←R(ADDR)	6	A(0)	D	R(ADDR)	
D←countup D	7		D	D	
A←shr A	8		A	A	
A←cil A	9		A	A	
A←A add R	10		A	A	R
A←A sub R	11		A	A	R
R←A	12		R	A	
C←D	13		C	D	
R←0	14	G′	R	0	
A←0	15		A	0	
H←0	16	G′	H	0	
C←0	17	G′	C	0	
D←0	18	G′	D	0	
G←0	19		G	0	

kalmazásának feladását jelenti, ha ilyen a későbbiekben előfordul. Egy másik út a vezérlőbitek számának csökkentésére az, hogy azokat a mikroműveleteket vonjuk össze egy csoportba, amelyek ugyanannak az órajelciklusnak különböző fázisaiban jöhetnek létre, egyetlen vezérlőbit alatt. Például amint a 3.1. táblázatból látható, az F(13) vezérlőbit alatt lehet csoportosítani a H←F(ADDR) mikroműveletet, amely a P(2) órajelfázisban és a C←D és F←MT(H) mikroműveleteket, mivel mindkettő a P(3) órajelfázisban jön létre. Ez a redukció ismét csak korlátozza az egyedi mikroműveletek későbbi felhasználását. A vezérlőbitek csökkentésének harmadik útja a dekódoló alkalmazása. Helyettesítsük a hat F(17—22) vezérlőbitet a három F(17—19) bittel, és rendeljünk ehhez a három bithez egy dekódolót, amint azt az alábbi dekódolóutasítás specifikálja:

Decoder, K(0—5)=F(17—19) (3.12)

A dekódoló hat kimeneti kapocspárja K(j), ahol j értéke 0 és 5 között változik. A hat új vezérlőjel K(0)∗P(1),...,K(5)∗P(1) lesz. Ez a redukció azért lehetséges, mert a hat mikroművelet közül csak egy fordul elő egyidőben. Azonban a dekódoló alkalmazása korlátozza, hogy a jövőben a fenti hat mikroműveletből egynél többet szimultán üzemben lehessen használni. Röviden: a vezérlőbitek redukálásának ára az, hogy csökken a mikroprogramozott vezérlés rugalmassága egy újraprogramozás esetén.

A kereskedelmi forgalomban levő, nagyméretű számítógépekben igen sok mikroművelet van, mivel a gépi utasítások száma is nagy. Emiatt a mikroprogramtár kapacitása is sokkal nagyobb a példában látottnál. Ezért a mikroprogram mérete és a rugalmas mikroprogramozásra vonatkozó korlátozások közötti kompromisszum érdekes tervezési problémát jelenthet.

3.3. Tárolt logikájú számítógép

Az előző fejezetben tárgyalt mikroprogramozott vezérlésű számítógép csak kiolvasható (read-only) mikroprogram- (vezérlő-) tárat tartalmaz. Ennek a járulékos, véletlen hozzáférésű mikroprogramtárnak az alkalmazásával nő a mikroprogramozott számítógép költsége. A sebesség rovására külön tár nélkül is megtartható a mikroprogramozott vezérlés, ha a főtár egy részét használják fel vezérlőtárként. Minthogy a főtár olvasható—írható típusú tároló, a most már a főtárban tárolt mikroprogram ugyancsak változtatható vagy kiterjeszthető. Az ilyen mikroprogramozott számítógépet *tárolt logikájú számítógépnek* nevezik.

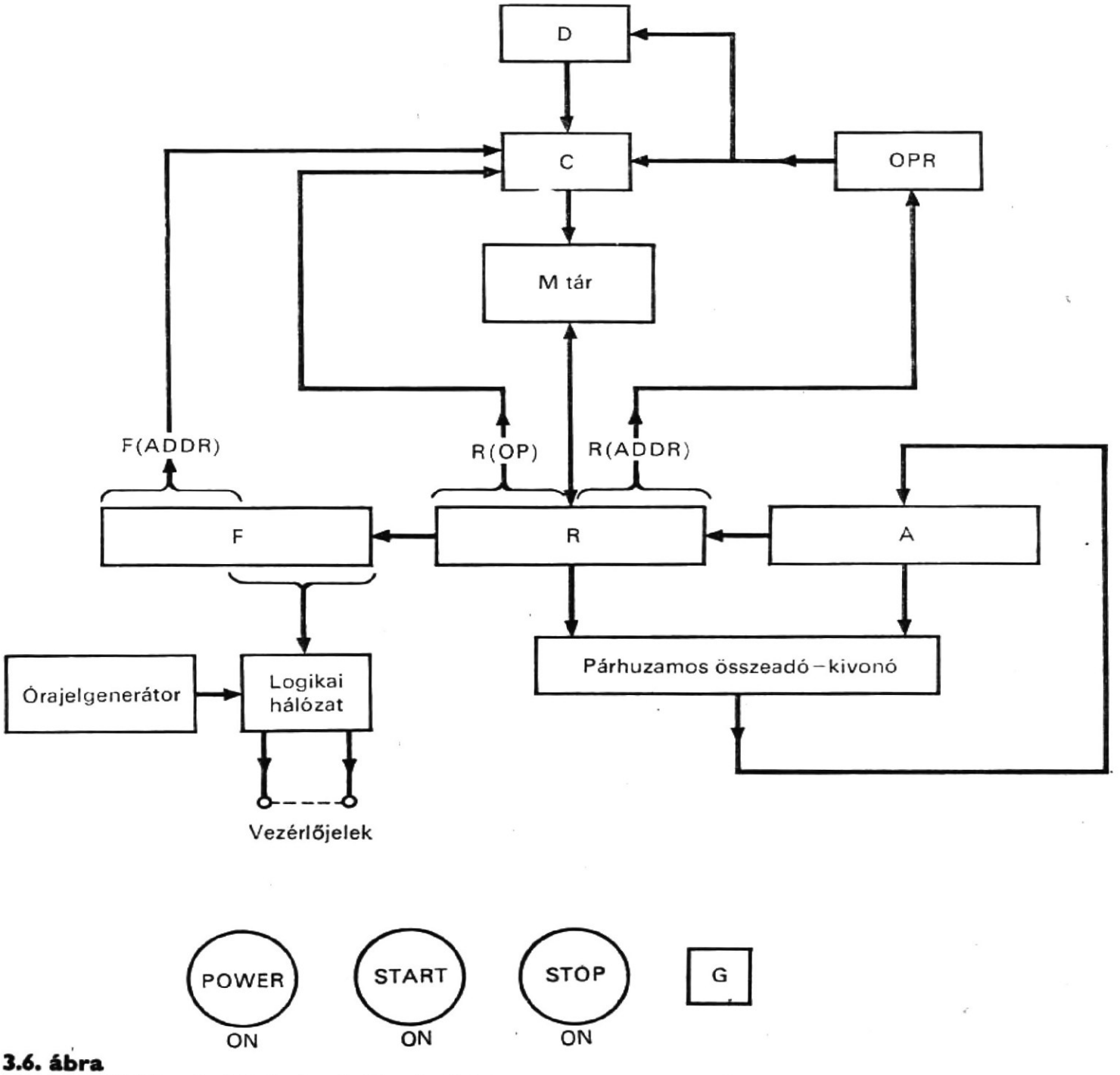
A tárolt logikájú számítógép illusztrálása érdekében elhagyjuk az előbbiekben leírt mikroprogramozott számítógép mikroprogram- (vezérlő-) tárát. Helyette a főtár első 1024 helyét használjuk vezérlőtárként. Tehát a főtár kétfajta utasítást fog tárolni: a „hagyományos" gépi utasítást és a mikroutasítást.

3.3.1.
Logikai hatásvázlat

A tárolt logikájú számítógép logikai hatásvázlata a 3.6. ábrán látható. Ezt a következő deklarációs utasítások írják le:

Comment, a tárolt logikájú számítógép struktúrája (3.13)

Register,	R(0—23),	$tár tartalomregisztere
	A(0—23),	$akkumulátor
	C(0—14),	$tár címregisztere
	D(0—14).	$utasításszámláló
	F(0—23),	$mikroutasítás-regiszter



3.6. ábra
Tárolt logikájú számítógép logikai hatásvázlata

```
                OPR(0—14),          $operandus címregiszter
                G,                  $start—stop vezérlési regiszter
Subregister,    R(OP)=R(0—5),
                R(ADDR)=R(9—23),
                F(ADDR)=F(0—9),
Memory,         M(C)=M(0—32767, 0—23),
Switch,         POWER(ON),
                START(ON),
                STOP(ON),
Clock,          P(1—6),
```

A fenti logikai hatásvázlat abban különbözik a (3.6) utasítások által leírt elrendezéstől, hogy nincs benne az MT mikroprogramtár és annak H címregisztere, de kiegészült az OPR operandus címregiszterrel és a háromfázisú órát hatfázisú óra váltotta föl.

3.3.2.
A műveletek sorrendjének leírása

Ennek a számítógépnek a mikroműveleteit öt típusra lehet osztani: azokra, amelyek kezdeti értéket adnak, indítanak, leállítanak, amelyek egy utasítást leírnak és amelyek végrehajtanak egy utasítást. A kezdetiérték-adó mikroműveletek a következők:

$$G \leftarrow 0,\ F \leftarrow 0,\ D \leftarrow 2000_8\ OPR \leftarrow 2000_8 \qquad (3.14)$$

Ha a G regiszter értéke 0, a számítógép „állj" állapotban van. Ha az F regiszter értéke 0, nem generál vezérlőjelet a vezérlőszó. A D és az OPR regiszterek értéke kezdetben 2000_8 és így az első utasítás a 2000_8 címen elhelyezett szó.

Az indítósorozat három lépésből áll:

```
1. lépés,   G←1, C←0,                              (3.15)
2. lépés,   IF (G=1) THEN (R←M(C)),
3. lépés,   F←R, C←OPR,
```

Ha a G regiszter értéke 1, a számítógép belép a „futási"állapotba. A C regiszter 0 értéket kap és megkezdődik a tárkiolvasási művelet. A 0 címen elhelyezett első mikroutasítást kiolvassuk a tárból és az az R regiszterbe kerül. A mikroutasítást ezután átvisszük az F regiszterbe, a C regiszter pedig a 2000_8 címet kapja meg, ahol az első gépi utasítást tárolják.

A leállási sorozat két lépésből áll:

```
1. lépés,   G←0,                                   (3.16)
2. lépés,   IF (G=0) THEN (F←0, D←2000₈, OPR←2000₈),
```

Ha a G regiszter értéke 0, az F, D és OPR regiszterek kezdeti értéket kapnak. A számítógép ismét kész az indításra.

Az utasításlehívási sorozat négy lépésből áll:

```
1. lépés,   R←M(C),
2. lépés,   C←R(OP), OPR←R(ADDR), D←countup D,     (3.17)
3. lépés,   R←M(C),
4. lépés,   F←R, C←OPR,
```

Két tárhozzáférés van az utasításlehívási sorozat alatt. Az első hozzáférés lehívja az

utasítást. Az utasítás helyét normális körülmények között a D regiszterben levő cím határozta meg, egyébként az OPR-ben levő cím, ha a számítógépet először indítják. A második hozzáférés azt a következő mikroutasítást hívja, amelynek helyét az első tárhozzáféréskor kapott utasítás műveletikód-része határoz meg. Ezután a mikroutasítás az F regiszterbe kerül.

Valamely utasításvégrehajtási sorozat legalább négy lépést tartalmaz:

```
1. lépés,    R←M(C),                                   (3.18)
2. lépés,    C←F(ADDR),
3. lépés,    R←M(C),
4. lépés,    F←R, C←D,
```

Ismét csak két tárhozzáférés van. Az első hozzáférés azt az operandust hívja, amelynek helyét az OPR regiszterben levő cím határozza meg. A második hozzáférés az előző mikroutasítás címrésze által lokalizált, következő mikroutasítást hívja. A mikroutasítást ezután az F regiszterbe kell átvinni. Ugyancsak át kell vinni a következő utasítás címét a D regiszterből a C címregiszterbe.

3.3.3. Vezérlőjelek

Minden óraciklus hat órajelfázisból áll. Két tárciklus van minden órajelciklusban: egy tárciklus a mikroutasítás hívására és egy az operandus hívására. Az alábbiakban ehhez a két híváshoz tartozó lépéseket a vezérlőjeleiket írjuk le:

```
/P(6)/    C←D vagy C←OPER, $tárciklus kezdete          (3.19)
/P(1)/    R←M(C),
/P(2)/                     $tárciklus vége
/P(3)/    C←F(ADDR) vagy   $a tárciklus kezdete
          C←R(OP),
/P(4)/    R←M(C),
/P(5)/                     $tárciklus vége
```

Az alábbiakban leírjuk egy szónak a tárba történő beírására használható mikroműveleteket és vezérlőjeleiket.

```
/P(6)/    C←OPR,           $a tárciklus kezdete
/P(1)/
/P(2)/    M(C)←R,          $a tárciklus vége
```

Amint a fentiekben látható, minden egyes tárciklus három órajelfázissal esik egybe.

3.3.4. A vezérlőszó formátuma

Miután meghatároztuk a kezdetiérték-adáshoz, indításhoz, leálláshoz, utasításlehíváshoz és utasításvégrehajtáshoz szükséges feladatokat, a következőkben a vezérlőszó formátumát kell megválasztani. Ebben a számítógépben a formátumot lényegében ugyanolyannak választjuk, mint amilyent a 3.1. táblázatban bemutattunk, eltekintve azoktól a változásoktól, amelyek azért váltak szükségessé, mert a főtár egy részét választottuk vezérlőtárnak. Ez a formátum a 3.4. táblázatban látható.

3.4. táblázat

Vezérlőszó formátuma

Vezérlőbit	Órajelfázis	Mikroműveletek
F(0—9)		következő mikroutasítás címe
F(10)	P(1)	R←M(C)
F(11)	P(3)	C←R(OP), D←countup D
F(12)	P(4)	IF (G) THEN (R←M(C), OPR←R(ADDR)) ELSE (F←0, D←2000_8, R(CIM)←2000_8)
	P(6)	F←R, C←OPR
F(13)	P(3)	C←F(ADDR)
	P(4)	R←M(C)
	P(6)	F←R, C←D
F(14)	P(1)	R←A
	P(2)	M(C)←R
F(15)	P(2)	A←A add R
F(16)	P(2)	A←A sub R
F(17)	P(1)	D←OPR
F(18)	P(1)	IF (A(0)) THEN (D←OPR)
F(19)	P(1)	A←shr A
F(20)	P(1)	A←cil A
F(21)	P(1)	A←0
F(22)	P(1)	G←0
F(23)		nincs felhasználva

3.3.5. A műveletek sorrendjének leírása

A 3.4. táblázatban található szóformátummal az 1.5. folyamatábrát a következő utasításokkal írhatjuk le:

Comment, a tárolt logikájú számítógép működésének leírása

Comment, kezdetiérték-adás

/POWER(ON)/ G←0, F←0, D←2000_8, OPR←2000_8,

Comment, start-művelet

/START(ON)*P(3)/ C←1, F(12)←1, C←0,

Comment, stop-művelet

/STOP(ON)/ G←0,

Comment, FETCH (utasításlehívás) lépései, ha C=0

/F(10)*P(1)/ R←M(C),
/F(11)*P(3)/ C←R(OP), D←countup D,
/F(12)*P(4)/ IF (G) THEN (R←M(C), OPR←R(ADDR))
ELSE (F←0, D←2000_8, OPR←2000_8),
/F(12)*P(6)/ F←R, C←OPR,

```
Comment, ADD művelet, ha C=1
C/F(10)*P(1)/       R←M(C),
/F(15)*P(2)/        A←A add R,
/F(13)*P(3)-        C←F(ADDR),
/F(13)*P(4)/        R←M(C),
/F(13)*P(6)/        F←R, C←D,

Comment, SUB művelet, ha C=2
C/F(10)*P(1)/       R←M(C),
/F(16)*P(2)/        A←A sub R,
C/F(13)*P(3)/       C←F(ADDR),
C/F(13)*P(4)/       R←M(C),
C/F(13)*P(6)/       F←R, C←D,

Comment, JOM művelet, ha C=3
/F(18)*P(1)/        IF (A(0)) THEN (D←OPR),
C/F(13)*P(3)/       C←F(ADDR),
C/F(13)*P(4)/       R←M(C),
C/F(13)*P(6)/       F←R, C←D,

Comment, STO sorozat, ha C=4
/F(14)*P(1)/        R←A,
/F(14)*P(2)/        M(C)←R,
C/F(13)*P(3)/       C←F(ADDR),
C/F(13)*P(4)/       R←M(C),
C/F(13)*P(6)/       F←R, C←D,

Comment, JMP művelet, ha C=5
/F(17)*P(1)/        D←OPR,
C/F(13)*P(3)/       C←F(ADDR),                    (3.21)
C/F(13)*P(4)/       R←M(C),
C/F(13)*P(6)/       F←R, C←D,

Comment, CIL sorozat, ha C=7
/F(20)*P(1)/        A←cil A,
C/F(13)*P(3)/       C←F(ADDR),
C/F(13)*P(4)/       R←M(C),
C/F(13)*P(6)/       F←R, C←D,

Comment, CLA sorozat, ha C=8
C/F(10)*P(1)/       R←M(C),
/F(21)*P(1)/        A←0,
C/F(15)*P(2)/       A←A add R,
C/F(13)*P(3)/       C←F(ADDR),
C/F(13)*P(4)/       R←M(C),
C/F(13)*P(6)/       F←R, C←D,

Comment, STP sorozat, ha C=9
/F(22)*P(1)/        G←0,
C/F(13)*P(3)/       C←F(ADDR),
C/F(13)*P(4)/       R←M(C),
C/F(13)*P(6)/       F←R, C←D,
                    END
```

3.3.6. Mikroprogram

A mikroprogramozott vezérlésű számítógép (3.10) leírásához hasonlóan a tárolt programú számítógép működésének sorrendi leírása is 13 csoportra oszlik. Az egyes csoportok élén egy-egy comment utasítás áll, és a csoportok valamelyik speciális műveletsorozat mikroműveleteit tartalmazzák. Minthogy az első három csoport mikroműveletei kézi vezérlésűek, nem jelennek meg a mikroprogramban. A maradék 10 csoport mikroműveleteit 10 mikroutasítással alakíthatjuk át, amelyek a mikroprogramot alkotják. A vezérlőszavak formátuma hasonló a 3.1. és 3.4. táblázatban levőkhöz, és a 3.2. táblázat mikroprogramja egyúttal a tárolt logikájú számítógép mikroprogramja is.

3.4. Mikroprogramozott vezérlés egy sorozatból a legnagyobb szám meghatározására

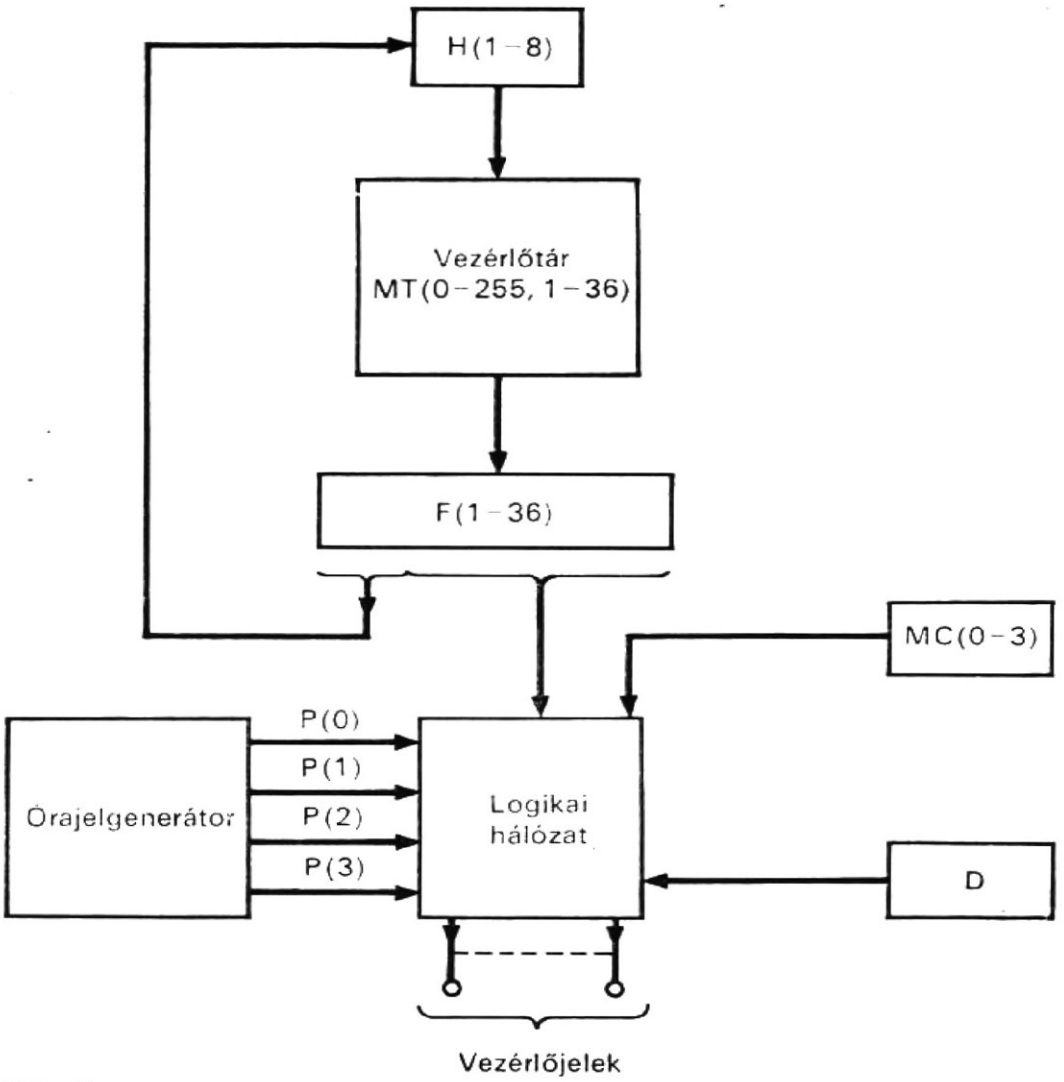
A mikroprogramozott vezérlésre utolsó példaként egy, a 2. fejezetből már ismert feladatot választunk. Célunk az *n* darab, előjel nélküli bináris számból a legnagyobb szám megkeresése. A logikai hatásvázlatot a 2.7. ábra mutatta be, az egyinás után végrehajtandó feladatokat a 2.8. folyamatábra mutatja, az utasítások leírása pedig a (2.12)-ben található.

3.4.1. Mikroprogramozott vezérlés logikai hatásvázlata

A mikroprogramozott vezérlés logikai hatásvázlata a 3.7. ábrán látható. Az MT mikroprogramtár kapacitása 256, egyenként 36-bites szó, és a H címregiszter és F tartalom-(mikroutasítás-) regiszter tartozik hozzá. A P(0—3) négyfázisú órajelgenerátor, a D egybites regiszter és az MC 4-bites regiszter állítja elő a vezérlőjeleket. A STATUS regiszterben állapotbit helyezkedik el. Az OLVAS regiszter kezdeményezi a tárból történő olvasás műveletét. A MEMSTART regiszter a tárműködést kezdeményezi, ha a RESET kapcsoló 1 értéken van. A RESET kapcsoló kezdeményezi a működést. A fentieket a következőképpen írhatjuk le:

Comment, a mikroprogramozott vezérlés struktúrája (3.22)

```
Register,   H(1—8),      $mikroprogramtár címregisztere
            F(1—36),     $mikroutasítás regisztere
            MC(0—3),     $főtárciklus regisztere
            D,           $tárciklus-várakozásregisztere
            STATUS,      $állapotbit
```



3.7. ábra
Mikroprogramozott vezérlés logikai hatásvázlata egy sorozatból a legnagyobb szám meghatározására

```
                OLVAS,              $tárolvasási művelet kezdeményezése
                MEMSTART,           $tárművelet indítása
Block,          ADDR(IF (STATUS) THEN (H←F(1—8)) ELSE (H←countup H)),
Memory,         MT(H)=MT(0—255,1—36),
Switch,         RESET(ON),          $start-kapcsoló
Clock,          P(0—3),             $négyfázisú órajelgenerátor

Comment, az MC regiszter gyűrűs számlálóként működik
/P(3)/          MC←cir MC
```

3.4.2.
Időzítés és a vezérlőjelek előállítása

Az egyes főtárciklusokat úgy választjuk meg, hogy a négy mikroprogram- (vezérlő-) tár ciklussal és mindegyik mikroprogramtár-ciklus egy óraciklussal essék egybe. Ezért minden mikroprogramtár-ciklusban négy lépés van és minden főtárciklusban 16 lépés. Ehhez a 16 lépéshez a vezérlőjeleket a következő, 16 cimkéből álló sorozat írja le, amelyekhez még az indítócimke és az MC ciklusregiszter cimkéje járul.

Comment, a cimkékkel leírt vezérlőjelek (3.23)

```
/RESET(ON)/  MC←8, MEMSTART←1,
/MC(0)*P(0)/                    $a főtár- és a mikroprogramtár-ciklus kezdete
/MC(0)*P(1)/
/MC(0)*P(2)/
/MC(0)*P(3)/                    $a mikroprogramtár-ciklus vége
/P(3)/       MC←cir MC
/MC(1)*P(0)/                    $a mikroprogramtár-ciklus kezdete
/MC(1)*P(1)/
/MC(1)*P(2)/
/MC(1)*P(3)/                    $a mikroprogramtár-ciklus vége
/P(3)/       MC←cir MC
/MC(2)*P(0)/                    $a mikroprogramtár-ciklus kezdete
/MC(2)*P(1)/
/MC(2)*P(2)/
/MC(2)*P(3)/                    $a mikroprogramtár-ciklus vége
/P(3)/       MC←cir MC
/MC(3)*P(0)/                    $a mikroprogramtár-ciklus kezdete
/MC(3)*P(1)/
/MC(3)*P(2)/
/MC(3)*P(3)/                    $a mikroprogramtár-ciklus vége
/P(3)/       MC←cir MC
```

A mikroprogramtár ciklusainak előző négy lépését a P(0—3) órajelgenerátor négy fázisa állítja sorrendbe, míg az egyes főtárciklusokban a négy mikroprogramtár-ciklust az MC(0—3) gyűrűs számláló négy állapota időzíti. A számlálót ciklikusan jobbra léptetik az egyes mikroprogramtár-ciklusok végén.

Egy-egy főtárciklus folyamán egy szó vagy kiolvasásra vagy beírásra kerül az X főtárból(ba). Tételezzük föl, hogy az átvitel a főtár C címregiszterébe, valamint a főtár olvasási vagy írási műveletének kezdeményezése a második lépés alatt történik. Az olvasási művelethez a szó az R tartalomregiszterben a hatodik lépés alatt áll rendelkezésre. Tehát:

```
/MC(0)*P(1)/  C←K, OLVAS←1,
/MC(1)*P(1)/  R←X(C),                                   (3.24)
```

A beírási művelethez a főtárban tárolandó szót átviszik a 12-ik lépés előtt az R tartalomregiszterbe. Tehát[7]:

```
/MC(0)*P(1)/  C←K, BEÍR←1,
/MC(2)*P(3)/  X(C)←R,                                   (3.25)
```

Bizonyos mikroműveletek minden tárciklusban előfordulnak. A mikroprogramtár vezérlőjeleit leíró cimkék a következők:

```
/P(0)*D'/                       $a mikroprogramtár-ciklus kezdete   (3.26)
/P(1)*D'/
/P(2)*D'/
/P(3)*D'/                       $a mikroprogramtár ciklus vége
```

Ha a D regiszter tartalma 0, akkor „megjelennek" a fenti cimkék; egyébként pedig eltűnnek. Tehát a D regiszter kapuzza a vezérlést, és ha D=1, akkor leállítja a mikroprogramtár ciklusaihoz tartozó vezérlőjeleket.

[7] Ehhez azonban a (3.22)-ben leírt struktúrában kellene egy BEIR regiszter is (hasonlóan az OLVAS regiszterhez), a tárba való beírási művelet kezdeményezésére. *(Fordító)*

Az egyes mikroprogramtár-ciklusok alatt egy mikroutasítás kerül kiolvasásra a mikroprogramtárból. Tételezzük föl, hogy a mikroprogramtár címének a H regiszterbe való átvitele és a mikroprogramtárból történő olvasás kezdeményezése egyaránt a megelőző mikroprogramtár-ciklus P(3) órajelfázisa alatt megy végbe, és a mikroutasítás az F tartalomregiszterben az aktuális mikroprogram- (rövidítve: mt) tár-ciklus P(0) órajelfázisa alatt áll rendelkezésre, vagyis

```
/P(3)*D'*F(15)/   DO ADDR        $a megelőző mt-ciklus vége
/P(0)*D'*F(9)/    F←MT(H),       $az aktuális ciklus kezdete
/P(1)*D'/
/P(2)*D'/
/P(3)*D'*F(15)/   DO ADDR,       $az aktuális mt-ciklus vége
```

ahol az F(9) és F(15) vezérlőbitek szerepét az alábbiakban magyarázzuk meg. Az F regiszterben levő mikroutasítás által aktivizált mikroműveletek az aktuális mikroprogramtár-ciklus P(1—3) órajelfázisai alatt kerülnek végrehajtásra, mivel a mikroutasítás hívása a P(0) fázis alatt történik.

3.4.3.
A mikroutasítás formátuma

A 3.5. táblázatban látható a mikroutasítás formátuma. Az F regiszter 36 bitje három csoportra oszlik: az F(1—8) mezőre, amely a mikroprogramtár-címet tartalmazza, az F(9—21) mezőre, amelyben a vezérlőbitek vannak és az F(23—26) mezőre, amely nincs kihasználva. Az F(9—22) mezőben valamennyi bit feladata a hozzátartozó órajelfázissal együtt egy vagy több mikroművelet vezérlése; tehát minden mikroműveletthez vagy mikroművelet-csoporthoz egy vezérlőjel tartozik.

3.5. táblázat
Mikroutasítás formátuma

Vezérlőbit	Órajelfázis	Mikroműveletek
F(1—8)		mikroprogramtárcím
F(9)	P(0)	F←MT(H)
F(10)	P(1)	C←0, MŰVKÉSZ←OFF, D←1
F(11)	P(1)	OLVAS←1 (vagy R←X(C))
F(12)	P(1)	K←R(ADDR)
F(13)	P(1)	J←K
F(14)	P(2)	D←0
F(15)	P(3)	IF (STATUS) THEN (H←F(1—8)) ELSE (H←countup H)
F(16)	P(1)	C←K
F(17)	P(1)	A←R
F(18)	P(1)	K←countdn K
F(19)	P(1)	IF (ATV(0)=1) THEN (A←R, J←K)
F(20)	P(3)	STATUS←0
F(21)	P(1)	IF (K=0) THEN (MŰVKÉSZ←BE, STATUS←1)
F(22)	P(1)	STATUS←1,
F(23—36)		nincs felhasználva

3.4.4.
A műveletek mikroprogramozása

A 3.5. táblázatban látható mikroutasítás-formátummal most már a 2.8. folyamatábra CDL utasításokkal leírható. A következő leírás két részből áll: a kezdetiérték-adásból és az összehasonlításból.

```
Comment, a legnagyobb egész szám meghatározása                    (3.28)

Comment, kezdetiérték-adó rész
/RESET(ON)/          MC←8, H←0, MEMSTART←1, MŰVKÉSZ←OFF,
                       STATUS←0,
                     MC←cir MC,

Comment, a főtárciklus kezdete                                    (3.29)
/P(0)*D'*F(9)/       F←MT(H),        $mikroutasítás lehívása (H=0)
/MC(0)*P(0)/
/MC(0)*P(1)*F(10)/   C←0, MŰVKÉSZ←OFF,
                     D←1,            $mt vezérlőjelek leállítása
/MC(0)*P(1)*F(11)/   OLVAS←1,
/MC(0)*P(2)/
/MC(0)*P(3)/
/MC(1)*P(0)/
/MC(1)*P(1)*F(11)/   R←X(C),
/MC(1)*P(2)/
/MC(1)*P(3)/
/MC(2)*P(0)/
/MC(2)*P(1)*F(12)/   K←R(ADDR),
/MC(2)*P(2)/
/MC(2)*P(3)/
/MC(3)*P(0)/
/MC(3)*P(1)*F(13)/   J←K,
/MC(3)*P(2)*F(14)/   D←0,            $mt vezérlőjelek indíthatók
/MC(3)*P(3)/
/P(3)*D'*F(15)/      DO ADDR,        $H←countup H

Comment, a főtárciklus kezdete                                    (3.30)
/P(0)*D'*F(9)/       F←MT(H),        $mikroutasítás lehívása (H=1)
/MC(0)*P(0)/
/MC(0)*P(1)*F(16)/   C←K,
/MC(0)*P(1)*F(11)/   OLVAS←1,
/MC(0)*P(2)/
/MC(0)*P(3)/
/P(3)*D'*F(15)/      DO ADDR         $H←countup H
/P(0)*D'*F(9)/       F←MT(H),        $mikroutasítás lehívása (H=2)
/MC(1)*P(0)/
/MC(1)*P(1)*F(11)/   R←X(C),         $X(c) lehívása
/MC(1)*P(2)/
/MC(1)*P(3)/
/P(3)*D'*F(15)/      DO ADDR         $H←countup H
/P(3)*D'*F(9)/       F←MT(H),        $mikroutasítás lehívása (H=3)
/M(2)*P(0)/
```

```
/MC(2)*P(1)*F(17)/    A←R,
/MC(2)*P(1)*F(18)/    K←countdn K,
/MC(2)*P(2)/
/MC(2)*P(3)/
/P(3)*D'*F(15)/       DO ADDR,        $H←countup H
/P(0)*D'*F(9)/        F←MT(H),        $mikroutasítás lehívása (H=4)
/MC(3)*P(0)/
/MC(3)*P(1)*F(21)/    IF (K=0) THEN (MŰVKÉSZ←ON, STATUS←1)
/MC(3)*P(2)/
/MC(3)*P(3)*F(20)/    STATUS←0,
/P(3)*D'*F(15)/       DO ADDR,        $H←F(1—8) (H=5 vagy 8)

Comment, összehasonlító rész                                          (3.31)
Comment, a főtárciklus kezdete

/P(0)*D'*F(9)/        F←MT(H),        $mikroutasítás lehívása (H=5)
/MC(0)*P(0)/
/MC(0)*P(1)*F(16)/    C←K,
/MC(0)*P(1)*F(11)/    OLVAS←1,
/MC(0)*P(2)/
/MC(0)*P(3)/
/P(3)*D'*F(15)/       DO ADDR,        $H←countup H
/P(0)*D'*F(9)/        F←MT(H),        $mikroutasítás lehívása (H=6)
/MC(1)*P(0)/
/MC(1)*P(1)*F(11)/    R←X(C),         $X(c) lehívása
/MC(1)*P(2)/
/MC(1)*P(3)/
/P(3)*D'*F(15)/       DO ADDR,
/P(0)*D'*F(9)/        F←MT(H),        $mikroutasítás lehívása (H=7)
/MC(2)*P(0)/
/MC(2)*P(1)*F(19)/    IF (ÁTV(0)=1) THEN (A←R, J←K),
/MC(2)*P(1)*F(18)/    K←countdn K,
/MC(2)*P(1)*F(22)/    STATUS←1,
/MC(2)*P(2)/
/MC(2)*P(3)*F(20)/    STATUS←0,
/P(3)*D'*F(15)/       DO ADDR,        $H←F(1—8) (H=4)
                      END
```

A leírásban a kezdetiérték-adó rész teszi ki csaknem az egész első két főtárciklust, míg az összehasonlító rész egy főtárciklust vesz igénybe egy-egy iterációhoz. Az első főtárciklus alatt a D regiszternek 1 értéket ad az MC(0)*P(1)*F(10) vezérlőjel. Amikor a D regiszter értéke 1, a vezérlőjelek generálása a mikroprogramtár-ciklusban leáll [l. a (3.26) utasítások cimkéit], míg a főtárciklusban folytatódik a vezérlőjelek generálása. Következő mikroutasítás lehívása addig nem történik a mikroprogramtárból, amíg el nem kezdődik a következő főtárciklus; ez időt ad a főtárhoz való hozzáféréshez. Ennek eredményeképpen a mikroutasítások az F regiszterben maradnak az első főtárciklusban. Az MC←cir MC mikroműveletet a P(3) órajelfázis vezérli, minthogy az minden egyes tárciklusban végbemegy.

3.4.5. Mikroprogram

A legnagyobb szám meghatározásához tartozó mikroprogram a 3.6. táblázatban található, és nyolc mikroutasításból áll. Ezeket a mikroutasításokat a következő CDL utasítás nyolcszori előfordulásának helyén tartalmazza a leírás:

/P(0) * D' * F(9)/ F ← MT(H),

amely egy mikroutasítást olvas ki a mikroprogramtárból. E nyolc mikroutasítás mindegyikét a (3.28) műveletsorozat leírásában egy mikroprogramtár-ciklusban levő végrehajtási utasítások írják le. Az egyes mikroutasítások kódolt formáját (1-eseit és

3.6. táblázat

A legnagyobb szám meghatározásának mikroprogramja

H	F(1—8)	F(9)	F(10)	F(11)	F(12)	F(13)	F(14)	F(15)	F(16)	F(17)	F(18)	F(19)	F(20)	F(21)	F(22)
0	000_8	1	1	1	1	1	1	1	0	0	0	0	0	0	0
1	000_8	1	0	1	0	0	0	1	1	0	0	0	0	0	0
2	000_8	1	0	1	0	0	0	1	0	0	0	0	0	0	0
3	000_8	1	0	0	0	0	0	1	0	1	1	0	0	0	0
4	000_8	1	0	0	0	0	0	1	0	0	0	0	1	1	0
5	000_8	1	0	1	0	0	0	1	1	0	0	0	0	0	0
6	000_8	1	0	1	0	0	0	1	0	0	0	0	0	0	0
7	004_8	1	0	0	0	0	0	1	0	0	1	1	1	0	1

Megjegyzés: H a mikroprogramtárcím.

0-it) ebből a leírásból a következőképpen kaphatjuk meg. Az 1-eseket azok a vezérlőbitek határozzák meg, amelyek értéke 1 valamely mikroprogramtár-ciklusban levő végrehajtási utasítás cimkéiben. Így az első mikroprogramtár-ciklus tartalmazza az első nyolc végrehajtási utasítást a (3.29) leírásban. A cimkékben a vezérlőbitek: F(9)...F(15); ezek azok a bitek, amelyeket 1-gyel jelöltünk a 3.6. táblázat első mikroutasításában. A második mikroutasítás-ciklus tartalmazza az első négy végrehajtási utasítást a (3.30) leírásban. A vezérlőbitek a cimkékben: F(9), F(11), F(15) és F(16); ezek azok a bitek, amelyek 1-gyel vannak jelölve a táblázat második mikroutasításában. A többi mikroutasítást hasonló módon kapjuk meg. Ezzel a módszerrel készült a 3.6. táblázatban található mikroprogram.

Feladatok

3.1. Szerkesszünk vezérlési ciklus diagramot (alakja a 3.4. ábrához hasonló) a legnagyobb számot meghatározó mikroprogramozott logikai hálózathoz!

3.2. Szerkesszük meg a (3.10) utasítással leírt mikroprogramozott számítógép vezérlési ciklus diagramját.

3.3. Legyen oktális 12 a műveleti kódja annak az utasításnak, amely az A regiszter tartalmát három bitpozícióval ciklikusan jobbra lépteti. Írjunk mikroutasítássorozatot ennek az utasításnak a mikroprogramozott számítógépben való implementálására!

3.4. Valamilyen programozási nyelven (például a Fortran, az Algol vagy a PL/1) írjuk le a (3.6) és (3.10) utasításokkal megadott mikroprogramozott számítógép működését!

3.5. A 3.3. táblázat tartalmazza a (3.6) és (3.10) utasításokkal leírt mikroprogramozott számítógép mikroműveleteit, a mikroutasítások öt mezőjével együtt. Feltételezve, hogy minden mikroműveletet egy mikroutasításnak tekintünk, készítsünk mikroprogramot ezen mikroutasítások felhasználásával, a mikroprogramozott számítógép implementálására! További mikroműveleteket (tehát mikroutasításokat) is fel lehet használni, ha szükségesek.

3.6. Szerkesszük meg a tárolt logikájú számítógép vezérlési ciklus diagramját.

3.7. Adjuk meg a 3.4. feladatot a (3.13) és (3.21) utasításokkal leírt tárolt logikájú számítógépre!

3.8. Javasoljunk egy olyan megoldást, amelynek segítségével védeni lehet a tárolt logikájú számítógépet a felhasználói programtól jövő hibás hozzáférés ellen!

3.9. Írjunk assembly nyelven programot a legnagyobb számnak a 2.6. folyamatábra szerinti meghatározására! Határozzuk meg azon főtárciklusok számát, amelyek az assembly nyelvű program végrehajtásához szükségesek! Hasonlítsuk össze az így nyert főtárciklusok számát azon főtárciklusok számával, amelyek a (3.28) utasításokkal leírt mikroprogram végrehajtásához szükségesek!

3.10. Használjunk a (3.22) utasításokkal leírtakhoz hasonló vezérlést és készítsünk mikroprogramot:
a) két, előjel nélküli szám soros összehasonlítására, ennek elvét a 2.4. folyamatábra mutatja be;
b) két, előjeles szám soros összehasonlítására, amelynek algoritmusát a 2.5. ábra mutatja be!

3.11. Használjunk a (3.22) utasításokkal leírtakhoz hasonló vezérlést és készítsünk mikroprogramot az első 1000 prímszám előállítására! Az algoritmust a 2.13. ábrán levő folyamatábra tartalmazza.

3.12. Tervezzünk egy vezérlési struktúrát és mikroprogramozzuk azt, ha a feladat a soros Gray—bináris kódkonverzió! Az algoritmust a 2.15. ábra illusztrálja.

3.13. Ismételjük meg a 3.12. feladatot bináris—decimális konverzióra, felhasználva a 2.18. folyamatábrát!

3.14. Ismételjük meg a 3.12. feladatot a Mercer-féle többszörös összeadásra, amelynek algoritmusát a 2.20. ábrán levő hatásvázlat mutatja be!

3.15. Ismételjük meg a 3.12. feladatot a teketalálatot számláló számítógépre, amelynek működését a 2.22. folyamatábrán mutattuk be!

Irodalom

[1] WILKES, M. V.: *The Best Way to Design an Automatic Calculating Machine.* Manchester University Computer Inaugural Conference, July, 1951, pp. 16...18.

[2] WILKES, M. V.—STRINDER, J. B.: *Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer.* Proceedings of the Cambridge Philosophical Society 49, Part 2, 1953, pp. 230...238.

[3] GLANTZ, H. T.: *A Note on Microprogramming.* Journal of the Association for Computing Machinery 3, No. 2, April, 1956, pp. 77...84.

[4] MERCER, R. J.: *Micro-programming.* J. of the ACM, April, 1957, pp. 157...171.

[5] WILKES, M. V.: *Microprogramming.* Proceedings of the Eastern Joint Computer Conference, December, 1958, pp. 18...20.

[6] BLANKENBAKER, J. V.: *Logically Microprogrammed Computers.* IRE Transactions on Electronic Computer EC-7, June, 1958, pp. 103...109.

[7] DINEEN, G. P.—LEBOW, I. L.—REED, I. S.: *The Logic of CG24.* Proceedings of the Eastern Joint Computer Conference, December, 1958, pp. 91...94.

[8] WILKES, M. V.—RENWICH, W.—WHEELER, D. J.: *The Design of the Control Unit of an Electronic Digital Computer.* Proceedings of the IEE (London), Vol. 105, Pt. B, June, 1959, pp. 121...128.

[9] KAMPE, T. W.: *The Design of a General-purpose Microprogram-controlled Computer with Elementary Structure.* IRE Transactions on Electronic Computers EC-9, June, 1960, pp. 208...213.

[10] SEMARNE, H. M.—POERTER, R. E.: *A Stored Logic Computer.* Datamation 2, No. 4, May, 1961, pp. 33...36.

[11] GRASELLI, A.: *The Design of Program-modifiable Microprogrammed Control Units.* IEEE Transactions on Electronic Computers EC-11, June, 1962, pp. 336...339.

[12] HAGIWARE, H.: *The KT Pilot Computer-a Microprogrammed Computer with a Phototransitor Fixed Memory.* Proceedings of AFIPS Congress, 1962, pp. 318...321.

[13] GERACE, G. B.: *Microprogrammed Control for Computing Systems.* IEEE Transactions on Electronic Computers, December, 1963, pp. 734...747.

[14] BOUTWELL, E., JR.—HOSKINSON, E. A.: *The Logical Organization of the PB 440 Microprogrammable Computer.* Proceedings of the FJCC, 1963, pp. 201...213.

[15] AMDAHL, L. D.: *Microprogramming and Stored Logic.* Datamation 10, 2, 1964. pp. 24...26.

[16] McGEE, W. C.: *The TRW-133 Computer.* Datamation 10, 2, 1964, pp. 27...29.

[17] HILL, R. H.: *Stored Logic Programming and Applications.* Datamation 10, 2, 1964, pp. 36...39.

[18] BECK, L.—KEELER, F.: *The C-8401 Data Processor.* Datamation 10, 2, 1964, pp. 33...35.

[19] BOUTWELL, E. O., Jr.: *The PB440 Computer.* Datamation 10, 2, 1964, pp. 30...32.

[20] STEVENS, W. T.: *The Structure of SYSTEM 360. Part II. System Implementation.* IBM System J. 3, 1964, pp. 136...143.

[21] TUCKER, S. G.: *Emulation of Large Systems.* Communications of the Association for Computing Machinery 8, No. 12, December, 1965, pp. 753...761.

[22] TUCKER, S. G.: *Microprogram for System/360.* IBM Systems Journal 6, No. 4, 1967. pp. 222...241.

[23] WEBER, H.: *A Microprogrammed Implementation of EULER on IBM System/360 Model 30.* Communications of the ACM, September, 1967, pp. 549...558.

[24] HUSSON, S.: *Microprogramming Manual for the 360 Model 50.* TR 00.1467, IBM System Development Division, October 2, 1967.

[25] FLYNN, M. J.—MacLAREN, M. D.: *Microprogramming Revisited.* Proceeedings of the ACM National Meeting, 1967, pp. 457...464.

[26] WILKES, M. V.: *The Growth of Interest in Microprogramming: A Literature Survey.* Computing Surveys, December, 1969, pp. 139...145.

[27] ROSIN, R. F.: *Contemporary Concepts of Microprogramming and Emulation.* Computing Surveys, December, 1969, pp. 197...212.

[28] HUSSON, S. S.: *Microprogramming: Principles and Practices.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1970.

[29] CHU, Y.: Introduction to Computer Organization. Englewood Cliffs, N. J., Prentice-Hall, Inc., 1970.

4. Fixpontos aritmetikai egység

Tárolt programú digitális számítógép aritmetikai utasításait az aritmetikai—logikai egységben hajtják végre. Ha az aritmetikai egység bináris számokat párhuzamosan ad össze és von ki (azaz a két szám összes bitjével egyszerre végez műveletet), akkor az párhuzamos bináris aritmetikai egység. A párhuzamos aritmetikai egységben van egy párhuzamos összeadó vagy egy párhuzamos kivonó, vagy lehet mindkettő is, az alkalmazott aritmetikai algoritmusoktól függően.

Ha bináris számokat előjelbittel és egy vagy több számjegybittel ábrázolják, és a bináris pont rögzítetten van elhelyezve valahol, akkor a bináris számokat *fixpontos bináris számoknak* nevezik. Ha a bináris számok exponenciális formában vannak kifejezve, amint azt a későbbiekben, a következő fejezetben leírjuk, akkor *lebegőpontos bináris számoknak* nevezik őket. A fixpontos számokat kezelő aritmetikai egységet fixpontos aritmetikai egységnek, míg a lebegőpontos számokat kezelő aritmetikai egységet *lebegőpontos aritmetikai egységnek* nevezik. Lehetséges természetesen olyan aritmetikai egység is, amely mind fixpontos, mind lebegőpontos számokat tud kezelni.

Ez a fejezet olyan párhuzamos aritmetikai egységet ír le, amely fixpontos bináris összeadást, kivonást, szorzást és osztást tud végrehajtani. Ezt az aritmetikai egységet először sorrendi logikai vezérléssel, majd ezután mikroprogram-vezérléssel is bemutatjuk. A következő fejezetben olyan párhuzamos, bináris aritmetikai egységgel ismerkedhetünk meg, amellyel lebegőpontos aritmetika valósítható meg.

4.1. Az aritmetikai egység struktúrája

Az aritmetikai egység leírása a fixpontos szám formátumával kezdődik, ezt követi a struktúra és a párhuzamos összeadó. Az összeadás, kivonás, szorzás és osztás végrehajtásához szükséges funkcionális elemeket későbbi részben mutatunk be.

4.1.1.
Fixpontos szám ábrázolása

Valamely bináris szám egyik ábrázolási módja a fixpontos formátumban történő ábrázolás. Ebben az ábrázolásban van egy előjelbit, egy vagy több számjegybit, és a bináris pont bármely két szomszédos bit között elhelyezkedhet. A bináris pontnak két, legelterjedtebben használt elhelyezkedése: a számot vagy egészként vagy törtszámként ábrázolja. A 4.1. ábra olyan fixpontos formátumot mutat, ahol az előjelbit a legértékesebb számjegybit pozíciójától balra helyezkedik el. Ha a bináris pont az előjelbit és a legértékesebb számjegybit között van, akkor az ilyen bináris ponttal rendelkező bináris számok törtszámok. Ha a bináris pont a legkisebb helyiértékű (legkevésbé értékes) számjegybittől jobbra helyezkedik el, akkor az így ábrázolt számok egészek.

A párhuzamos, bináris, fixpontos aritmetikai egységhez a fixpontos szám formátuma a 4.1. ábrán látható. Ebben 35 számjegybit van, továbbá 1 előjelbit. Tehát összesen 36 bitet tartalmaz. A számot egész számként ábrázoljuk. A bináris számok előjelét az előjelbit reprezentálja. Ha az előjelbit 0, a bináris szám pozitív; egyébként pedig negatív. Valahányszor negatív nulla fordul elő, azt rendszerint pozitívra állítják át. Megjegyezzük, hogy nincs olyan hardware, amely közvetlen megvalósítja a bináris pontot; a bináris pontot inkább közvetve valósítják meg, az aritmetikai algoritmusban.

4.1.2.
Struktúra

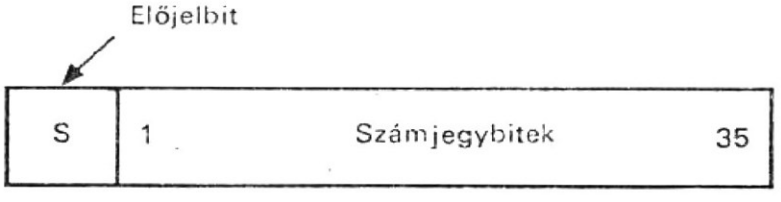
A párhuzamos, bináris, fixpontos aritmetikai egység négy regisztere:

- az AC akkumulátor;
- az MQ, szorzó—hányados (multiplier-quotient) regiszter,
- a TR tár tartalomregiszter;
- az LSZ lépésszámláló

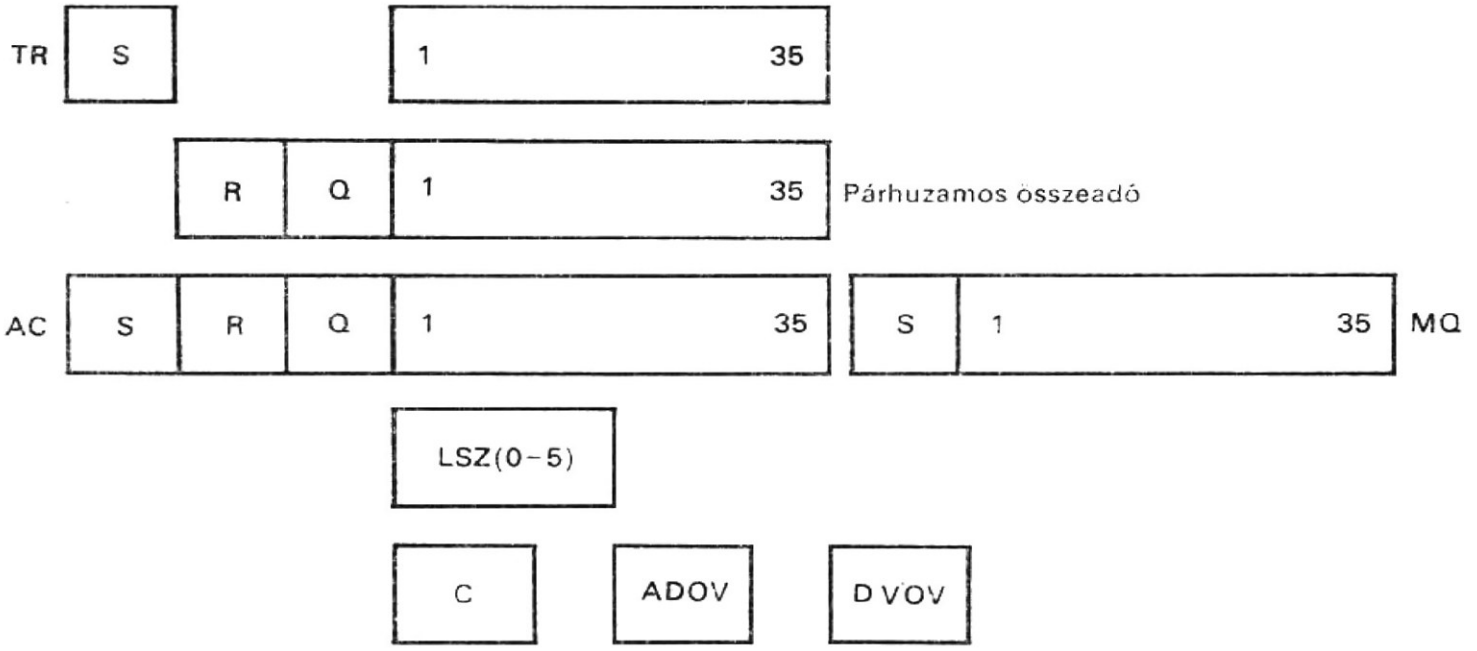
regiszterek. További funkcionális elem még a párhuzamos összeadó és a három egybites indikátor: a C, az ADOV és az DVOV. Ezek struktúrája a 4.2. ábrán látható. Az AC, MQ és TR regiszter mindegyike két részre oszlik: az egyik az előjelet, a másik a mennyiséget ábrázolja. Pontosabban: az AC(S), MQ(S) és TR(S) részregiszterek tárolják az előjelbiteket és az AC(M), MQ(M) és TR(M) részregiszterek tárolják a számjegybiteket. Az LSZ regisztert számlálóként használják a szorzás és osztás közben. A C regisztert a szorzóbitek módosításának vezérlésére használják a szorzás közben. Az ADOV és DVOV regiszterek az összeadás, ill. osztás közben bekövetkező túlcsordulást jelzik.

Comment, a fixpontos aritmetikai egység struktúrája (4.1)

```
Register,    AC(S,R,Q,1—35)     $akkumulátor
             TR(S,1—35),        $a tár tartalomregisztere
             MQ(S,1—35),        $szorzó—hányados regiszter
```



4.1. ábra
Fixpontos szám formátuma



4.2. ábra
Párhuzamos, bináris, fixpontos aritmetikai egység funkcionális elemei

```
                LSZ(0—5),          $lépésszámláló
                C,                 $vezérlőbit a szorzás alatt
                ADOV,              $összeadási túlcsordulás kijelző (addition over-
                                    flow indicator)
                DVOV,              $osztási túlcsordulás kijelző (division over-
                                    flow indicator)
Subregister,    AC(M)=AC(1—35),    $az AC regiszter számjegybitjei
                TR(M)=TR(1—35),    $a TR regiszter számjegybitjei
                MQ(M)=MQ(1—35),    $az MQ regiszter számjegybitjei

Comment, a párhuzamos összeadó leírása                                    (4.2)
Terminal,       ADD(R,Q,1—35)=ADTR(R,Q,1—35) ⊙ADAC(R,Q,1—35) ⊙
                   ⊙C(R,Q,1—35),
                C(R,Q,1—34)=ADTR(Q,1—35) * ADAC(Q,1—35)+
                   + ADAC (Q,1—35) * C(Q,1—35)+ C(Q,1—35) * ADTR(Q,1—35),
                C(35)=0,

Comment, a Z kapocspárok deklarációja                                     (4.3)
Terminal,       Z(R,Q,1—35)=0—0—AC(M) add2 0—0—TR(M)—0,

Comment, az add2 operátor leírása                                         (4.4)
Operator,       W(R,Q,1—35)=X(R,Q,1—35) add2 Y(R,Q,1—36),
Terminal,       C(35)=Y(36),
                C(R,Q,1—35)=X(Q,1—35) * Y(Q,1—35)+
                              +Y(Q,1—35) * C(Q,1—35)+
                              +C(Q,1—35) * X(Q,1—35),
/begin/         W(R,Q,1—35)=X(R,Q,1—35) ⊙Y(R,Q,1—35) ⊙C(R,Q,1—35),
                end of operator
```

A bemutatott utasításokban az AC regiszter tartalmazza az AC(Q) bitet, amely az AC regiszter előjelbitje és a legnagyobb helyiértékű számjegybit között helyezkedik el. Az AC(Q) bit a legnagyobb helyiértékű számjegybitből származó átvitelt vagy átvételt tárolja összeadás vagy kivonás alatt. Amint látni fogjuk, ezt a bitet kell majd meg-

vizsgálni annak meghatározására, hogy keletkezett-e túlcsordulás összeadás, vagy alulcsordulás kivonás után. Az AC regiszter egy másik bitet is tartalmaz, amelynek elnevezése AC(R) és amely az AC(Q) bitből érkező átvitelt tárolja szorzás közben. Látni fogjuk, ez a külön AC(R) bit azért kell, mert egy kétbites szorzási algoritmust alkalmazunk.

A párhuzamos összeadót, a Z kapocspárokat és az add2 operátort a következőkben írjuk le.

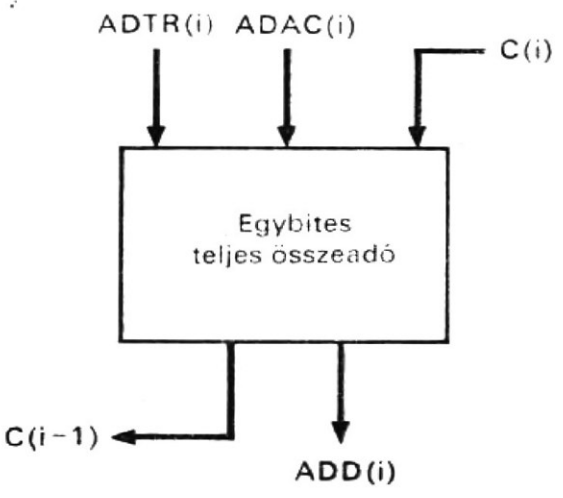
4.1.3. Párhuzamos összeadó

A párhuzamos összeadó szimultán adja össze két bináris szám azonos helyiértékű bitjeit. Rendszerint egybites teljesösszeadókból áll, amely mindegyikének három bemenete (egyik összeadandó bit, másik összeadandó bit és bemenet—átvitelbit) és két kimenete (az összegbit és a kimenet—átvitelbit) van. Legyen a bemenetek és a kimenetek elnevezése a 4.3. ábrán bemutatott, ily módon ADAC(i) az i-edik helyiérték egyik összeadandó bitje, ADTR(i) az i-edik helyiérték másik összeadandó bitje, C(i) az i-edik átvitelbit, C(i−1) az (i−1)-edik átvitelbit és ADD(i) az i-edik helyiérték összegbitje, tehát i a párhuzamos összeadó i-edik szakaszát reprezentálja. Ezt a teljesösszeadót a következő Boole-függvényekkel lehet leírni:

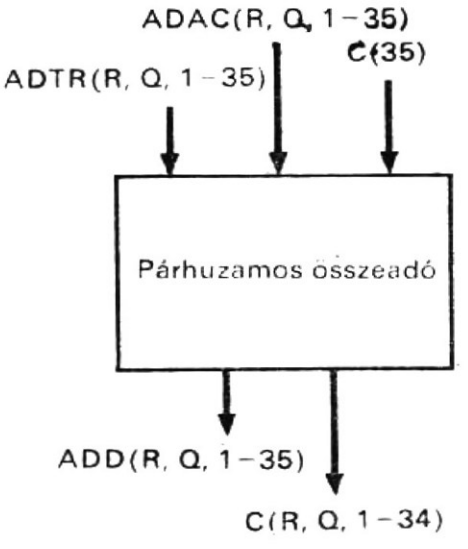
$$ADD(i) = ADTR(i) \odot ADAC(i) \odot C(i) \tag{4.5}$$

$$C(i-1) = ADTR(i) * ADAC(i) + ADAC(i) * C(i) + C(i) * ADTR(i) \tag{4.6}$$

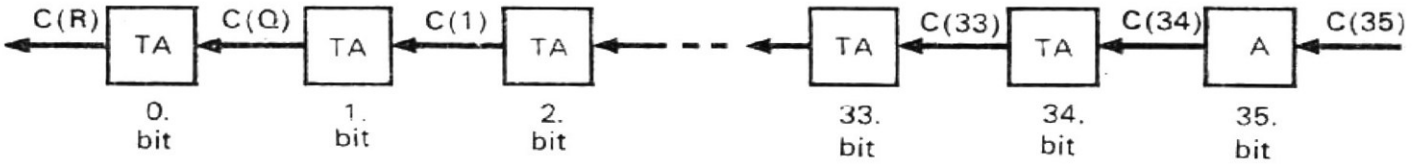
A fixpontos aritmetikai egységhez tartozó párhuzamos összeadó bemeneteinek és kimeneteinek elnevezései a 4.4. ábrán láthatók. Az ADAC(R,Q,1−35) és ADTR(1−35) általában az AC(R,Q,1−35), ill. a TR(1−35) regiszterek kimeneteihez kapcsolódnak. Az ADTR(R,Q) rendszerint nem a TR regisztertől kap vezérlést, hanem helyette 0-kból vagy 1-esekből álló konstansokat iktatnak be ebbe a két helyiértékbe, ha ez szükséges. A C(35), amely általában 0, a teljesösszeadó jobb oldali legszélső szakaszának bemeneti átvitele. Az ADD(R,Q,1−35) kimenetek az összegbitek. A C(R,Q,1−35) kimenetek a 37 teljesösszeadókból származó kimeneti átvitelek, azonban rendszerint nem tekintik őket tényleges kimeneteknek.



4.3. ábra
Egybites teljes összeadó (a párhuzamos összeadó *i*-edik tagja)



4.4. ábra
A párhuzamos összeadó bemeneteinek és kimeneteinek jelölései



4.5. ábra
Párhuzamos összeadó

Ehhez az aritmetikai egységhez a párhuzamos összeadót 37 egybites teljesösszeadó felfűzésével állítottuk elő. Ez a párhuzamos összeadó a 4.5. ábrán látható, ahol az egyszerűség kedvéért csak az átviteleket és a teljesösszeadókat (jelölésük TA) tüntettük fel. Ezt a párhuzamos összeadót írtuk le a (4.2)-ben a terminálutasításokkal (a kimeneti kapocspárokhoz tartozó logikai egyenletek megadásával).

4.1.4.
A Z terminálok

Az osztási algoritmusnak a későbbiekben bemutatandó leírásában vizsgálatot végzünk a párhuzamos összeadó egyik kimeneti terminálját. A vizsgálat során a párhuzamos összeadó egyik bemeneti halmaza az AC(M) részregiszterből, a másik bemeneti halmaz pedig a TR(M) részregiszterből áll. A párhuzamos összeadó Z-nek nevezett kimeneti terminálját a (4.3)-ban levő terminálutasítás írja le.

4.1.5.
Az add2 operátor

Az előzőekben leírt párhuzamos összeadó kényelmes alkalmazása érdekében definiálunk egy operátort, amelyet *add2*-nek nevezünk. Általában ezt az operátort arra alkalmazzák, hogy két, 37-bites bináris szám számjegy bitjeit adja össze; ebben az esetben a bemeneti átvitel C(35)=0. Esetenként arra használják, hogy egy kivonandó 2-es komplemensét hozzáadja egy kisebbítendőhöz, a kivonás összeadással történő helyettesítéseként; ebben az esetben a bemeneti átvitel C(35)=1. Ezért a bemeneti átvitelt a párhuzamos összeadó további bemenetének lehet tekinteni és az add2 operátort erre a célra hoztuk létre. (Ha nincs szükség valamely szám 2-es komplemensének egy másik számhoz történő hozzáadására, az *add* bázisoperátor is teljesen megfelelő erre a célra.) Az add2 operátort a (4.4)-ben leírt utasításokkal definiáltuk.

Az add2 operátor alkalmazásakor mind a W, mind az X terminálok 37 bitből állnak, míg az Y terminál 38-bites. Az Y(36) terminál reprezentálja a C(35) bemeneti átvitel értékét.

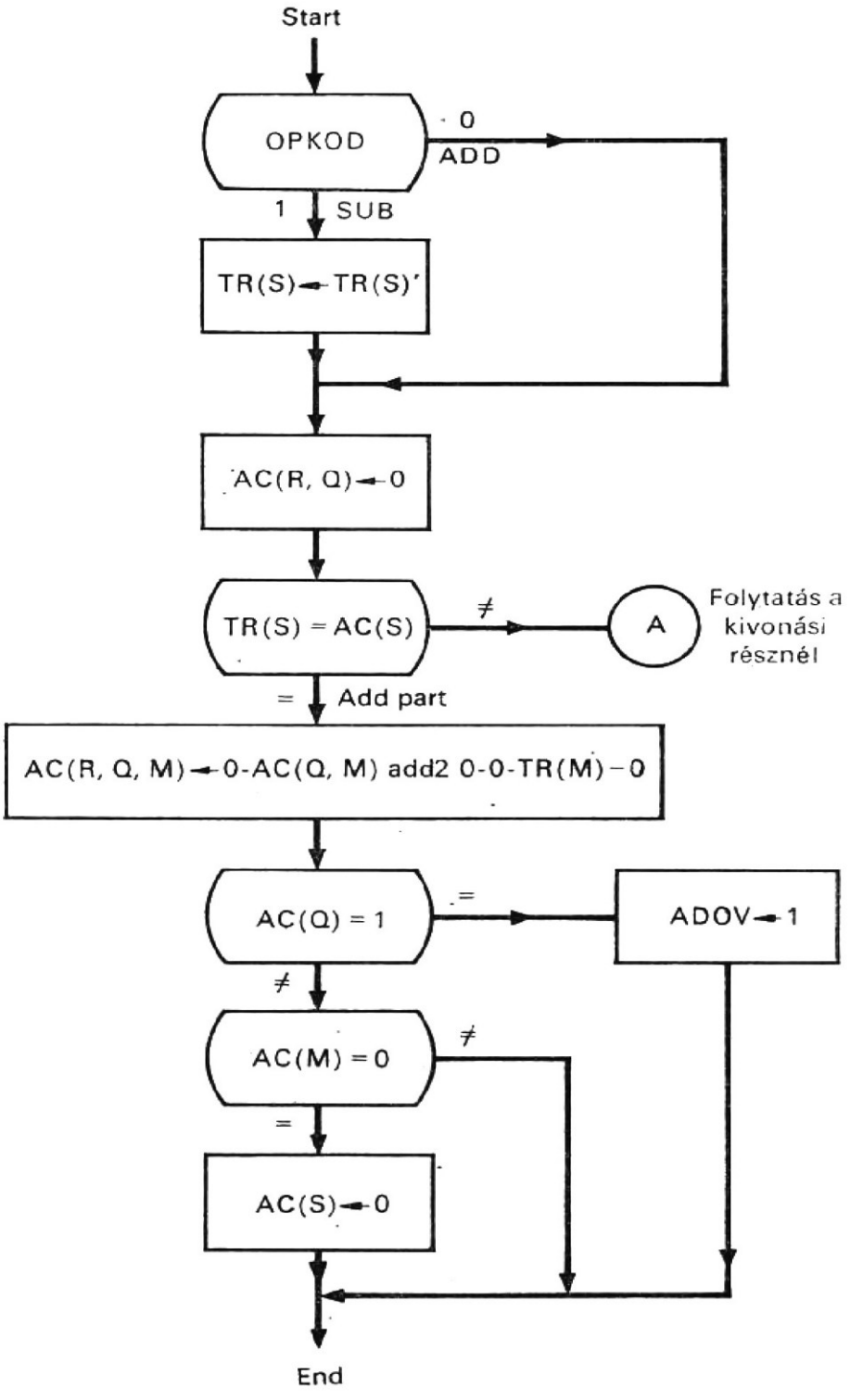
4.2.
Aritmetikai algoritmusok

Amint az előzőekben említettük, az aritmetikai egység előjellel és abszolút értékkel ábrázolt fixpontos bináris számok bináris összeadását, kivonását, szorzását és osztását tudja elvégezni. Az alkalmazott aritmetikai algoritmusokat folyamatábrákkal írjuk le.

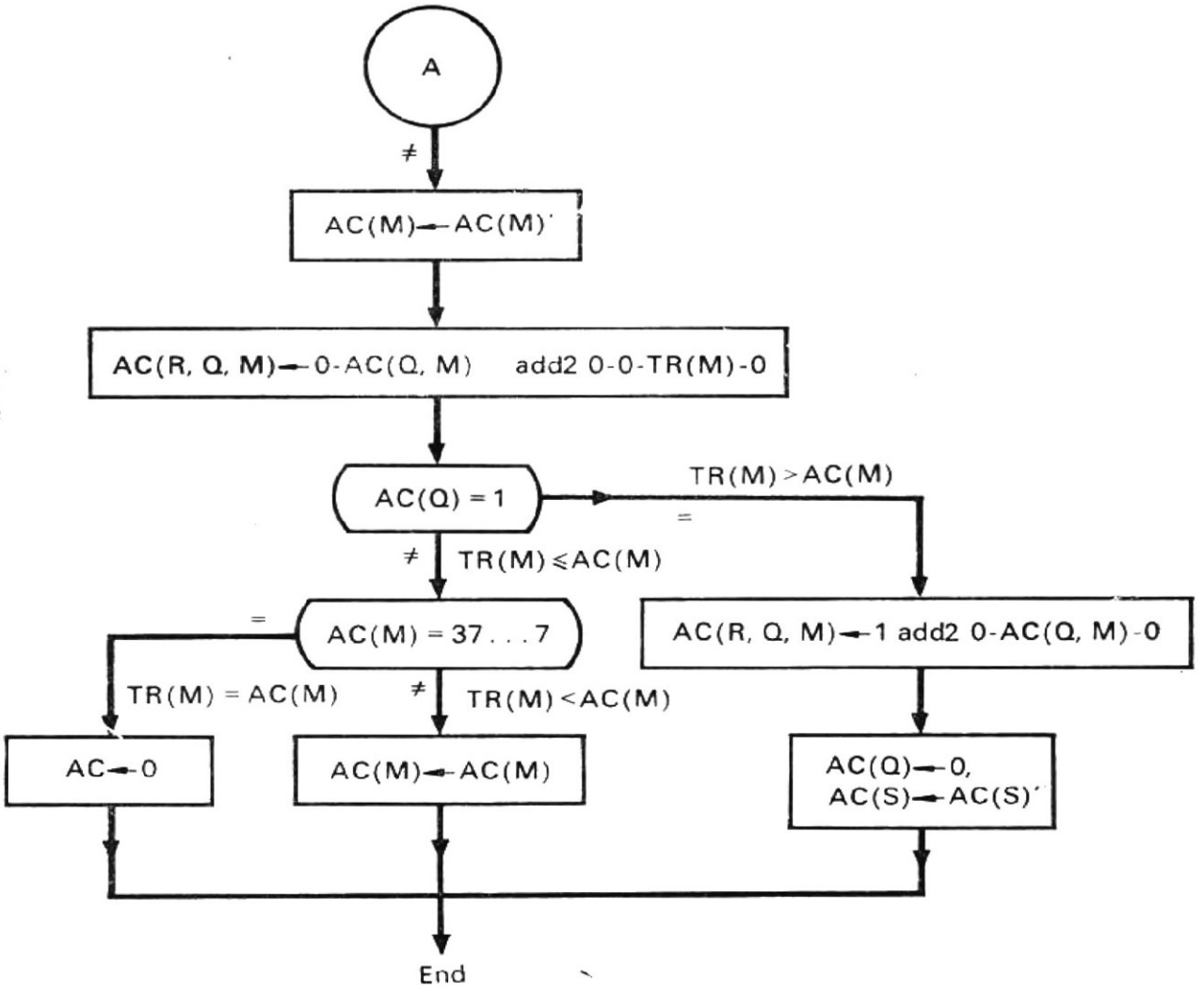
4.2.1.
Fixpontos összeadás és kivonás

A fixpontos összeadás hozzáadja a TR regiszterben levő fixpontos összeadandót az AC regiszterben levő fixpontos összeadandóhoz, míg a fixpontos kivonás kivonja a TR regiszterben levő fixpontos kivonandót az AC regiszterben levő fixpontos kisebbítendőből. A fenti két művelet során az MQ regiszterre nincs szükség.

A fixpontos összeadási és kivonási algoritmusokat a 4.6. és 4.7. folyamatábrák szemléltetik. Az algoritmust három részre lehet bontani:



4.6. ábra
Folyamatábra előjellel és abszolút értékkel ábrázolt bináris számok fixpontos összeadásának és kivonásának kezdetiérték-adó részéhez és összeadási részéhez



4.7. ábra
Folyamatábra előjellel és abszolútértékkel ábrázolt bináris számok fixpontos összeadásának és kivonásának kivonási részéhez

— kezdetiérték-adásra;
— összeadásra és
— kivonásra.

A kezdetiérték-adást a 4.6. ábra mutatja. Először is megváltoztatja a TR regiszterben levő kivonandó előjelét kivonás esetén, hogy a kivonás később összeadásra legyen visszavezethető. Ezután az AC(R,Q) biteket nullára állítja és összehasonlítja a két összeadandó előjelét, hogy meghatározza, vajon azonosak-e. Ha igen, a folyamatábra az összeadás részben; egyébként a kivonás részben folytatódik.

Az összeadás rész is látható a 4.6. ábrán. Ebben a részben a TR(M) részregiszterben elhelyezkedő összeadandó mennyiségét hozzáadják az AC(Q,M) részregiszterben levő összeadandó mennyiségéhez, és az eredményt az AC(Q,M) részregiszterben tárolják. Az előjel az AC regiszterben változatlan marad. Annak meghatározására, hogy előfordult-e túlcsordulás, megvizsgálják az AC(Q) bitet. Ha az 1-est tartalmaz, túlcsordulás van és az ADOV regiszter a túlcsordulás jelzésére 1 értéket kap. Ezzel az összeadási folyamat befejeződik. Ha az AC(Q) 0-t tartalmaz, továbbiakban megvizsgálandó, hogy nulla-e az AC(M) részregiszter mennyiség része. Ha nulla, az AC(S) bitbe nullát írnak, hogy negatív nulla az akkumulátorban ne fordulhasson elő.

A kivonási rész a 4.7. ábrán látható. Itt a kivonás az AC(M) részregiszterben levő kivonandó egyes komplemensének a hozzáadásával történik. Amint a 4.7. ábrán látható, az AC(M) részregiszterben levő mennyiségnek először az 1-es komplemens képzése megy végbe és azután ez hozzáadódik a TR(M) részregiszterben levő mennyiséghez, majd az eredményt az AC(R,Q,M) részregiszter tárolja. Ezután az AC(Q) bit vizsgálata

következik. Ha 1-est tartalmaz, az nem túlcsordulást jelent, csak azt mutatja, hogy a TR(M) részregiszterben levő mennyiség nagyobb, mint az AC(M) részregiszterben. Ebben az esetben az AC(Q) ciklikusan hozzáadódik az akkumulátorhoz, ezután AC(Q) bit 0 értéket kap, majd az előjelbit az akkumulátorban komplementálódik. Ha az AC(Q) bit 0-át tartalmaz, nincs ciklikus átvitel; ez azt jelzi, hogy a TR(M) részregiszterben levő érték vagy egyenlő vagy kisebb, mint az AC(M) részregiszter értéke. Ezután megvizsgálják az AC(M) részregisztert, nem tartalmaz-e csupa 1-est. Ha csupa 1-est tartalmaz, a TR(M) részregiszterben levő érték egyenlő az AC(M) részregiszterével és az eredménynek pozitív nullának kell lennie; ezért az AC regiszter 0 értéket kap. Ha viszont nem tartalmaz csupa 1-est, a TR(M) részregiszterben levő mennyiség kisebb, mint az AC(M) részregiszteré és az AC(M) részregiszterben elhelyezkedő eredmény 1-es komplemens alakban van. Ekkor a helyes eredmény úgy áll elő, hogy AC(M)-nek 1-es komplemensét vesszük. Ezzel az összeadási folyamat befejeződött.

A CDL-lel való leírásnak két változata lehetséges: a nemprocedurális és a procedurális [10]. A fenti összeadás és kivonás procedurális leírása közvetlenül felírható a 4.6. és 4.7. ábrából, és a következőkben mutatjuk be.

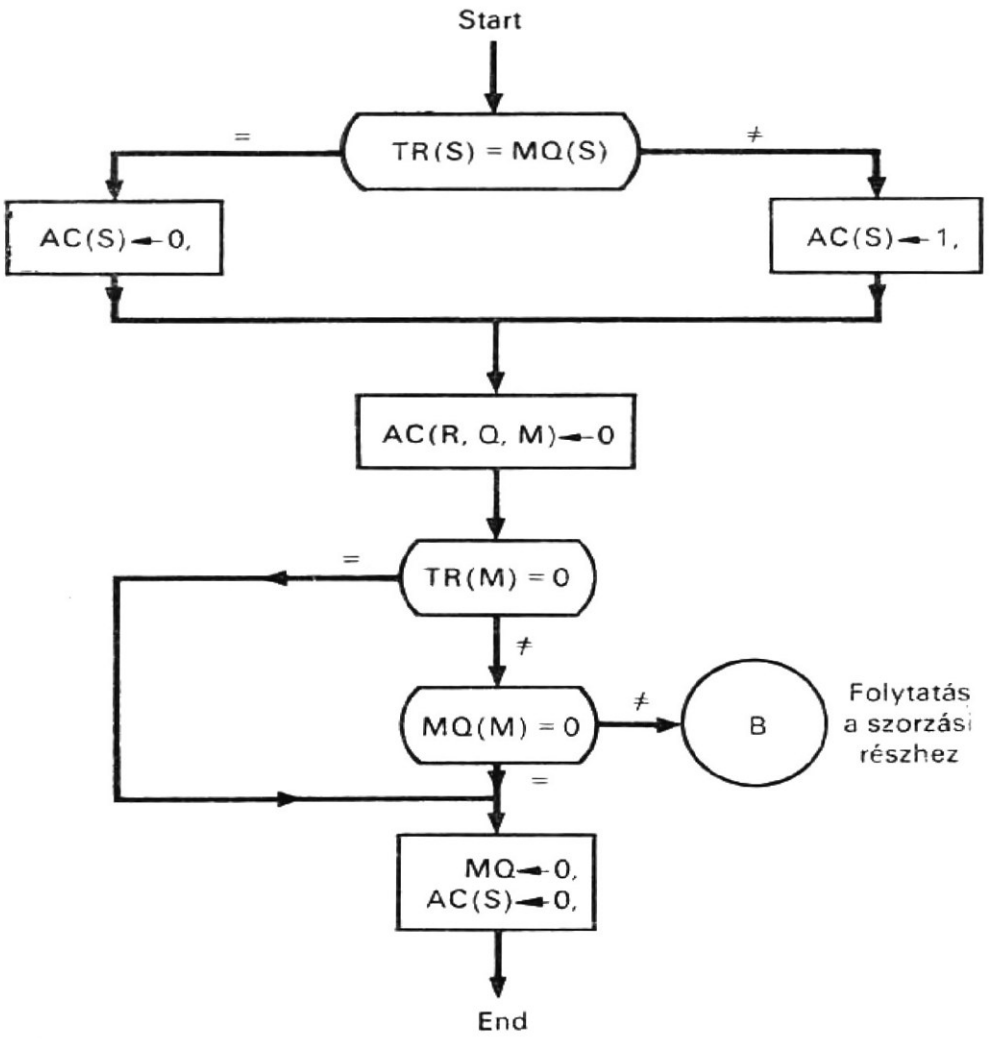
```
Comment, összeadási és kivonási műveletsorozat                    (4.7)
                IF ((OPKOD=0)+(OPKOD=1)) THEN (GOTO A0)
                  ELSE (GOTO A2);
/A0/            IF (OPKOD=1) THEN (TR(S)←TR(S)');
                AC(R,Q)←0, IF (TR(S)≠AC(S)) THEN (GOTO A);
                AC(R,Q,M)←0—AC(Q,M) add2 0—0—TR(M)—0;
                IF (AC(Q)=1) THEN (ADO←1, GOTO A2);
                 IF (AC(M)=0) THEN (AC(S)←0, GOTO A2);
/A/             AC(M)←AC(M)';
                AC(R,Q,M)←0—AC(Q,M) add2 0—0—TR(M)—0;
                IF (AC(Q)≠1) THEN (GOTO A1);
                AC(R,Q,M)←1 add2 0—AC(Q,M)—0;
                AC(Q)←0, AC(S)←AC(S)', GOTO A2;
/A1/            IF (AC(M)=37...7) THEN (AC←0) ELSE (AC(M)←AC(M)');
/A2/            END
```

Amint későbbiekben látni fogjuk, a fenti OPKOD egy kétbites regiszter, amely az utasítások műveleti kód (operációs kód) részét tartalmazza. Ha a műveleti kód 0, 1, 2, ill. 3, akkor az rendre összeadást, kivonást, szorzást ill. osztást jelöl.

Az előbbi procedurális leírás nem tartalmaz sem ütemezést, sem vezérlőjeleket. Az egy vagy több mikroutasítást megvalósító végrehajtási utasítások pontosvesszővel fejeződnek be. Az utasítások sorrendje a sorozat műveleteinek sorrendje. Az A0, A, A1 és A2 szimbolikus cimkéket önkényesen választottuk hivatkozási okokból, és GOTO utasítást használunk a cimkékre való hivatkozásra, a sorrendi működés rendjének megváltoztatásához.

4.2.2. Fixpontos szorzás

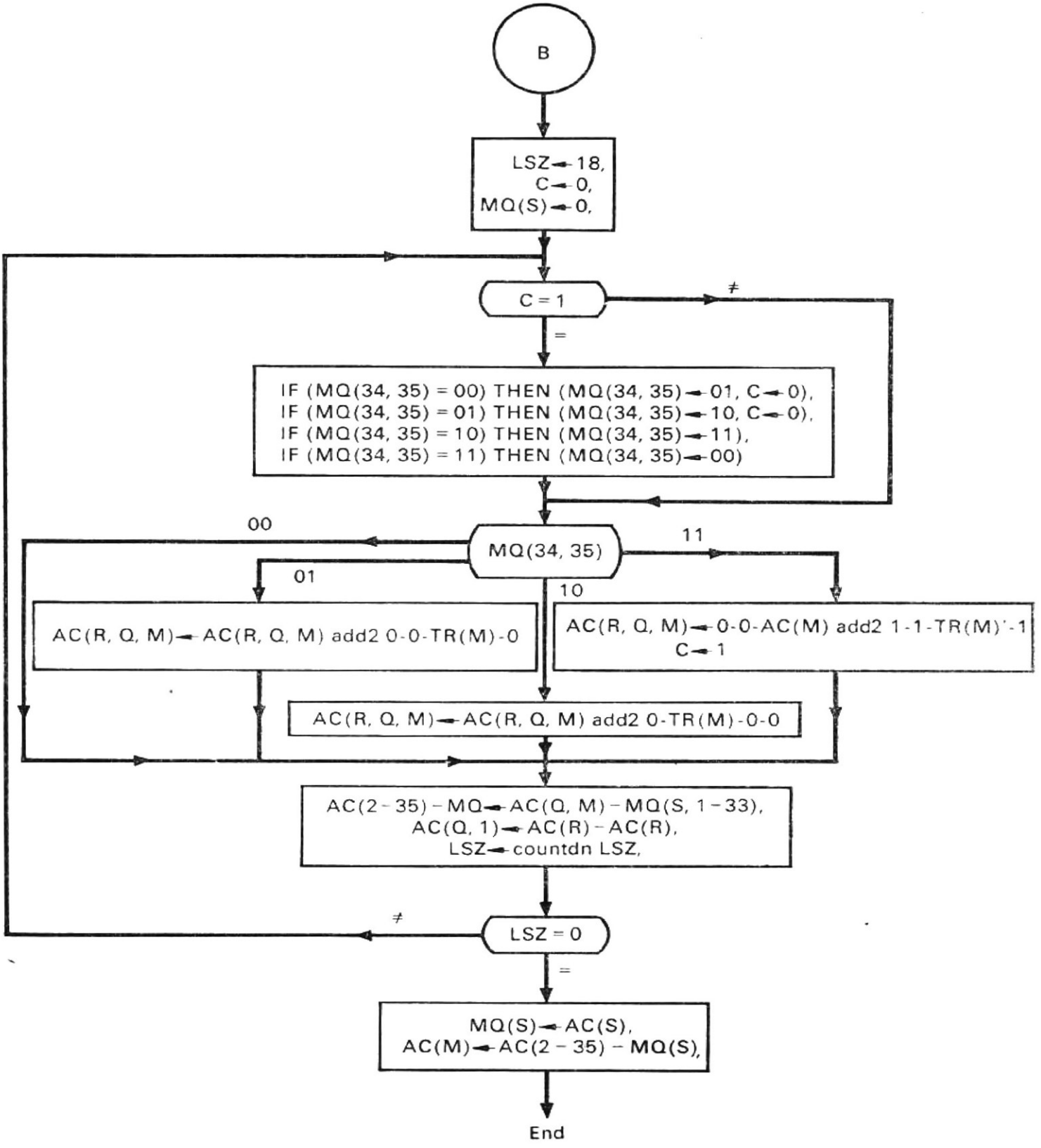
A fixpontos szorzás az MQ regiszterben levő fixpontos szorzót összeszorozza a TR regiszterben elhelyezkedő fixpontos szorzandóval, és a létrehozott szorzatot az AC és az MQ regiszterekből alkotott regiszterláncban helyezi el. A szorzat nagyobb helyiértékű része az AC regiszterben, míg a kisebb helyiértékű része az MQ regiszterben tárolódik. A szorzandó a TR regiszterben marad, az MQ regiszterben levő szorzó viszont elvész.



4.8. ábra
Folyamatábra előjellel és abszolút értékkel ábrázolt bináris számok fixpontos szorzásának kezdetiérték-adó részéhez

A 4.8. és 4.9. folyamatábrák előjellel és abszolút értékkel ábrázolt bináris számok fixpontos szorzását mutatja. A szorzási algoritmus két részre osztható: a kezdetiérték-adó részre (4.8. ábra) és a szorzási részre (a 4.9. ábra). A kezdetiérték-adó részben kerül összehasonlításra a szorzandó és a szorzó előjele. Ha ezek egyformák, az AC(S) bit nulla értéket kap; különben 1-re állítják. Ezután az AC regiszter R, Q, M részeit nulla értékre állítják. Ezt követően a szorzandó, ill. szorzó értékét vizsgálják 0-ra. Ha a TR(M) részregiszterben levő szorzandó nulla, akkor a szorzó az MQ(M) részregiszterben és AC(S) nulla értéket kap. Ha nem nulla a szorzandó, akkor megvizsgálják nullára az MQ(M) részregisztert. Ha az nulla, az AC és MQ regiszterekben a szorzatnak ismét nullának kell lennie; ezért csak az AC(S) és MQ(S) biteket kell nullára állítani és a szorzási folyamat befejeződött. Egyébként pedig a folyamatábra folytatódik a szorzási részben.

A szorzási részt a 4.9. folyamatábra mutatja. A szorzás az ismételt összeadás elvén történik, egyidőben két szorzóbit vizsgálatával. 35 bit szóhosszúságú szorzó esetén csak 18 összeadási ciklus szükséges. Amint a 4.9. ábrából látható, kezdetben az LSZ lépésszámláló 18 értéket kap, a C vezérlőregisztert pedig nulla értékre állítják. Továbbmenve az MQ(S) bitet nullára állítják és szorzó mennyiségi bitjének tekintik. Így a szorzó hossza 36 bit lesz. Ily módon a szorzási sorozat végén szükséges léptető mikroműveletek egyszerűsíthetők. A szorzás ezután az MQ(34,35) bitből álló, két utolsó helyiérték vizsgálatával kezdődik, amelyek értéke 00, 01, 10 vagy 11 lehet. Ha a két utolsó helyiértéke 00, nem kell semmit hozzáadni az akkumulátorhoz. Ha értékük 01, a szorzandó hozzáadódik az akkumlátorhoz. Ha 10, a szorzandó 1 bitpozícióval balra lép és így adódik hozzá az akkumulátorhoz. Ha 11, a szorzandó háromszori össze-

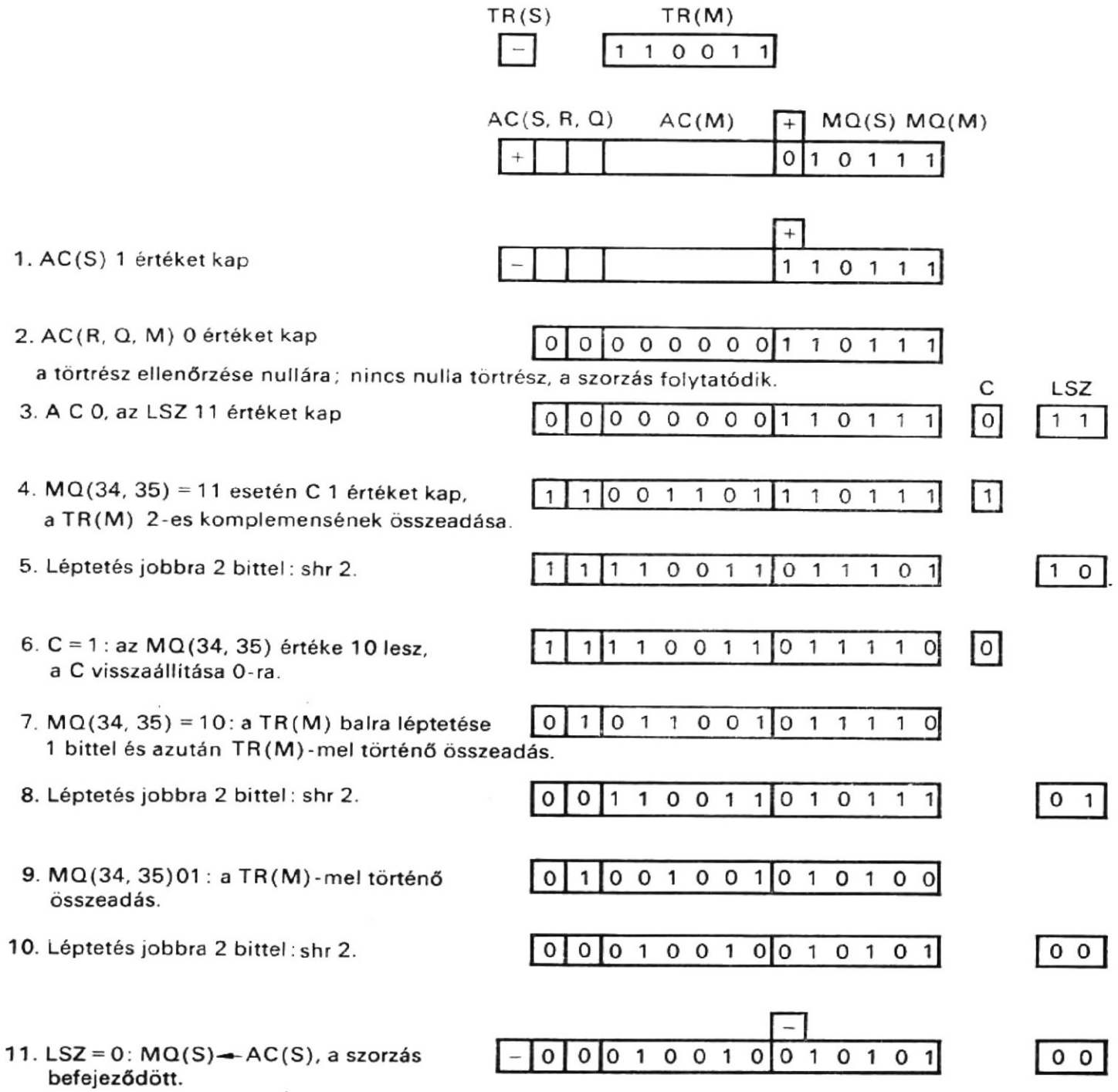


4.9. ábra
Folyamatábra előjellel és abszolút értékkel ábrázolt bináris számok fixpontos szorzásának szorzási részéhez

adása szükséges. Ahelyett, hogy egymás után háromszor adjuk össze a szorzandót, célszerűbb négyszer összeadni és egyszer kivonni. A szorzandó négyszeri összeadása egyenértékű azzal, hogy eggyel növeljük a szorzó következő két értékesebb bitjét a soron következő összeadási ciklus alatt; erre az emlékeztet, hogy a C értékét 1-re állítják. A szorzandó egyszeri kivonása a szorzandó 2-es komplemensének a hozzáadásával történik; ez olyan részszorzatot ad, amely negatív is lehet. A szorzandó 2-es

komplemensének a hozzáadása a szorzandó 1-es komplemensének az összeadásával végezhető, de ekkor ezzel egyidőben a C(35) bemeneti átvitel 1 értéket kap.

Miután a szorzandót nem adták hozzá, hozzáadták az akkumulátorhoz vagy kivonták az akkumulátorból, az AC(R,Q,M)—MQ(S,M) regisztersort két bitpozícióval jobbra léptetik és az LSZ lépésszámlálót 1-gyel csökkentik. Ennek a jobbra léptetésnek az aritmetikai értelmű léptetésnek kell lennie, minthogy a részszorzat lehet előjeles abszolút értékű [ezt az jelzi, hogy az AC(R) értéke 0], vagy 2-es komplemens kódú [ekkor az AC(R) értéke 1]. Ezután a lépésszámlálónak 0-ra való vizsgálata következik. Ha a tartalma nem 0, a C regiszter ellenőrzése következik 1-re. Ha a regiszter értéke 1, ez azt jelenti, hogy az MQ(34,35) szorzóbitek korrekcióját el kell végezni. Ez a korrekció az MQ(34,35) biteket a 00, 01, 10 és 11 értékekről rendre a 01, 10, 11 és 00 értékekre változtatja, és ugyanakkor, ha a változás 00-ról 01-re vagy 01-ről 10-re történt, a C regiszter értékét 0-ra állítja. Ismét megvizsgálják az MQ(34,35) biteket.



4.10. ábra
Előjellel és abszolút értékkel ábrázolt bináris számok fixpontos szorzása

Ennek megfelelően a szorzandó hozzáadása, nem hozzáadása vagy kivonása megy újból végbe; a regisztersor ismét két bitpozícióval jobbra lép; az LSZ lépésszámlálót újból eggyel csökkentik és azután megvizsgálják nullára. Ha nem 0, megismétlődnek a szorzóbiteket korrigáló és a további lépések, egészen addig, amíg az LSZ lépésszámláló értéke 0 nem lesz. Ekkor az AC(M) részregisztert 1 bitpozícióval balra léptetik és a szorzat előjelét beiktatják az MQ(S) bitbe. Ezzel a szorzás befejeződött. Megjegyezzük, hogy az MQ(34,35) bit a tizennyolcadik alkalommal csak 00 vagy 01 lehet, mivel az MQ(S)-t kezdetben 0-ra állították.

A 4.8. és 4.9. folyamatábrákon leírt szorzási eljárás sorrendi műveleteinek illusztrálására bemutatunk egy példát a 4.10. ábrán, a regiszterekben végbemenő lépésenkénti változásokra. Példánkban a következő mennyiségek szerepelnek:

szorzandó = 110011 = −51
szorzó = +010111 = +23
szorzat = −010010010101 = −1173

Figyeljük meg a 4.10. ábrán a jobbra léptetést, amely a bináris számok aritmetikai értelmű léptetésének szabályát követi a 2-es komplemens ábrázolásban.

A folyamatábra helyett a szorzási folyamatot procedurális módon is le lehet írni. A procedurális leírás a következő:

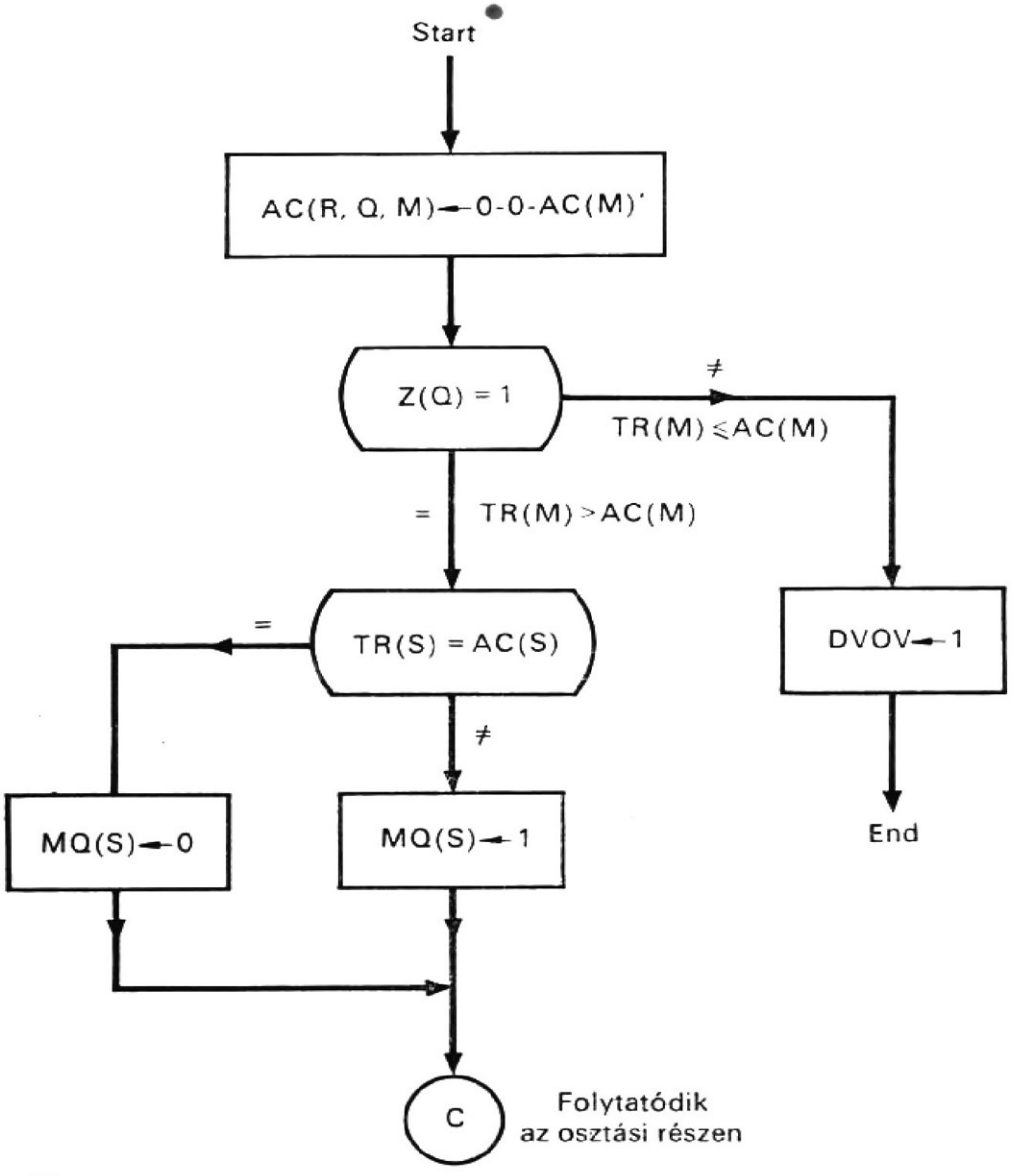
Comment, szorzási sorozat (4.8)

```
                IF (TR(S)=MQ(S)) THEN (AC(S)←0) ELSE (AC(S)←1);
                AC(R,Q,M)←0;
                IF (TR(M)=0) THEN (GOTO B0);
                IF (MQ(M)≠0) THEN (GOTO B);
/B0/            MQ←0, AC(S)←0, GOTO B2;
/B/             LSZ←18, C←0, MQ(S)←0;
/B1/            IF (C=1) THEN (IF(MQ(34,35)=00) THEN (MQ(34,35)←01,C←0),
                                IF (MQ(34,35)=01) THEN (MQ(34,35)←10,C←0),
                                IF (MQ(34,35)=10) THEN (MQ(34,35)←11),
                                IF (MQ(34,35)=11) THEN (MQ(34,35)←00));
                IF (MQ(34,35)=01) THEN (AC(R,Q,M)←AC(R,Q,M) add2
                  0−0−TR(M)−0,
                IF (MQ(34,35)=10) THEN (AC(R,Q,M)←AC(R,Q,M) add2
                  0−TR(M)−0−0),
                IF (MQ(34,35)=11) THEN (C←1,AC(R,Q,M)←0−0−AC(M)
                  add2 1−1−TR(M)'−1);
                AC(2−35)−MQ←AC(Q,M)−MQ(S,1−33), AC(Q,1)←AC(R)−
                  −AC(R);
                LSZ←countdn LSZ;
                IF (LSZ≠0) THEN (GOTO B1);
                MQ(S)←AC(S);
                AC(M)←AC(2−35)−MQ(S);
/B2/            END
```

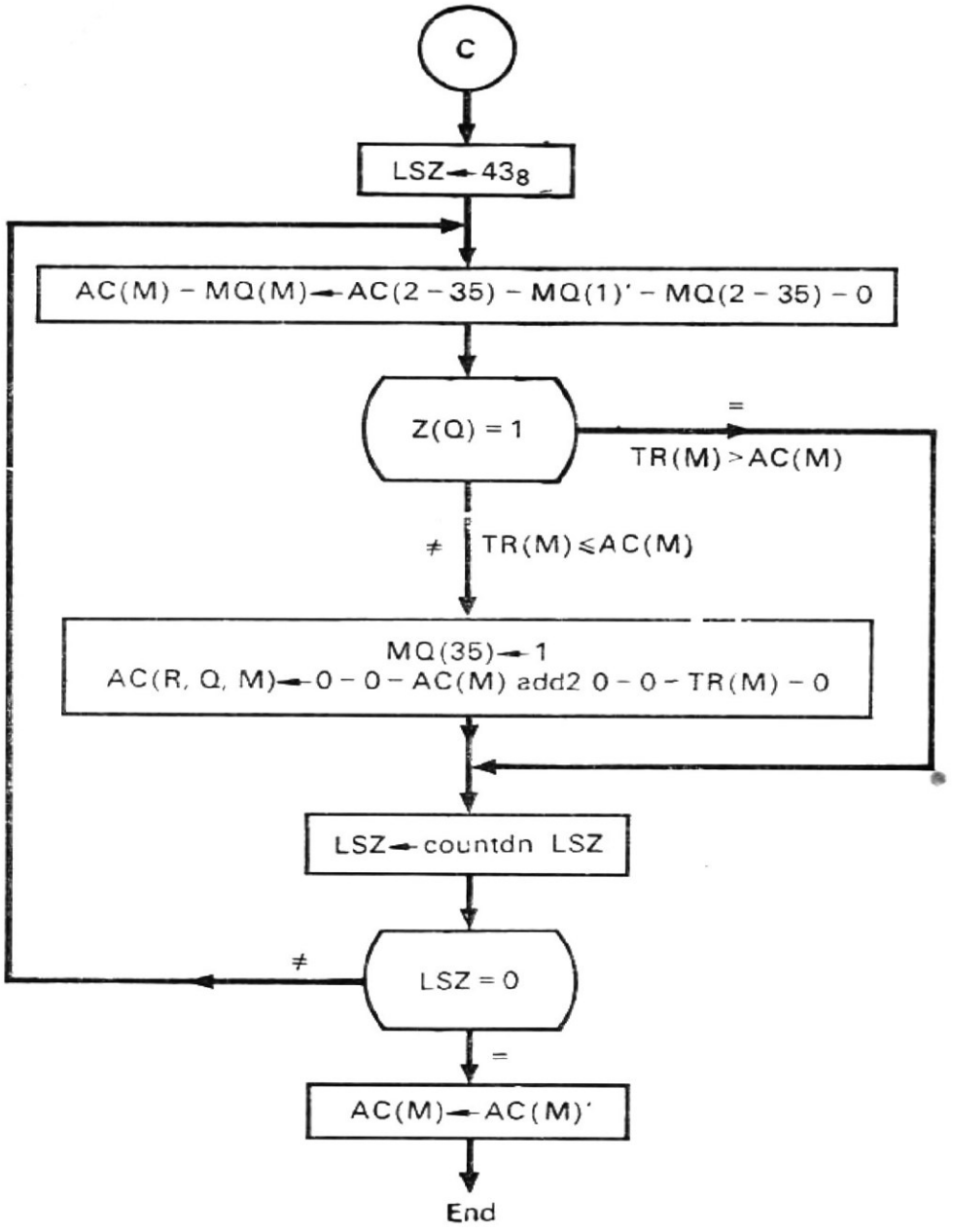
4.2.3.
Fixpontos osztás

A fixpontos osztás elosztja az AC és MQ regiszterekből összekapcsolt regisztersorban levő fixpontos osztandót a TR regiszterben elhelyezkedő fixpontos osztóval és fixpontos hányadost hoz létre az MQ regiszterben, valamint maradékot az AC regiszterben. Az osztó a TR regiszterben marad, míg az osztandó elvész a regisztersorban. Az osztás összehasonlítási módszert alkalmaz.

A 4.11. és 4.12. ábrán előjelekkel és abszolút értékkel ábrázolt bináris számok fixpontos osztásának folyamatábráit láthatjuk. A folyamatábrák két részre oszthatók: a 4.11 ábrán látható kezdetiérték-adó részre és a 4.12. ábrán levő osztási részre. A kezdetiérték-adó részben vizsgálják meg az osztáskor keletkező túlcsordulást és határozzák meg a hányados előjelét. Az osztási túlcsordulást kivonási teszttel határozzák meg. Ennek során kivonják az AC(M) részregiszterben levő osztandót a TR(M) részregiszterben elhelyezkedő osztóból. A kivonási tesztben a kivonás ténylegesen az AC(M) részregiszterben levő osztandó 1-es komplemensének hozzáadásával jön létre. Emlékezzünk vissza, hogy a (4.3) utasítást deklaráltuk a párhuzamos összeadó összegterminálјainak (kapocspárjainak) definiálására, 0—0—AC(M) és 0—0—TR(M), vala-



4.11. ábra
Folyamatábra előjellel és abszolút értékkel ábrázolt bináris számok fixpontos osztásának kezdetiérték-adó részéhez



4.12. ábra
Folyamatábra előjellel és abszolút értékkel ábrázolt bináris számok fixpontos osztásának osztási részéhez

mint C(35)=0 bemenetekkel. Ez a terminálutasítás lehetővé teszi, hogy a Z(Q) összegbitre hivatkozzunk az osztási túlcsordulás tesztben. Ha a kivonási teszt azt mutatja, hogy az összeg Z(Q) bitje 0, akkor ez azt jelzi, hogy az AC(M) részregiszterben levő osztandó nagyobb vagy egyenlő, mint a TR(M) részregiszterben levő osztó. Ezért osztási túlcsordulás jön létre, és a DVOV regiszterbe 1-et kell írni a túlcsordulás jelzésére; az osztási folyamat ezzel befejeződött. Ha a kivonási teszt azt mutatja, hogy a Z(Q) összeg kimenet 1, a kezdetiérték-rész folytatódik a hányados előjelének meghatározásával. A hányados előjele 0, ha a TR(S) és AC(S) bitek, vagyis az előjelek azonosak, egyébként pedig 1. Ezután a folyamatábra folytatódik az osztandónak az osztóval való osztásával.

Az osztási részben az AC(M) részregiszterben levő osztási részmaradék 1-es komplemens alakban marad. Az osztás az LSZ lépésszámlálónak oktális 43-ra (vagy decimális 35-re) való állításával kezdődik. Ezután az AC(M)—MQ(M) regisztersort egy bitpozícióval balra léptetik; a léptetés alatt az MQ(1) komplementálódik és csak utána lép balra, ezzel az AC(35) bitbe lép be; ily módon az AC(M) részregiszterben a részmaradék 1-es komplemens alakban marad. Összehasonlítják a TR(M) részregiszterben levő osztót az AC(M)-ben levő részmaradékkal. Ha az összehasonlítás azt mutatja, hogy

a Z(Q) kimenet 0, akkor az AC(M) részregiszterben levő részmaradék nagyobb vagy egyenlő, mint a TR(M)-ben levő osztó. Ebben az esetben az MQ(35) bit 1 értéket kap, és ugyanakkor a TR(M) részregiszterben az osztó hozzáadódik az AC(M) részregiszterben levő részmaradékhoz. Ha a Z(Q) kimenet 1, azt jelzi, hogy a TR(M) részregiszterben az osztó nagyobb, mint az AC(M) részregiszterben a részmaradék, ezért nem kell összeadást végezni. Ezután 1-gyel csökken az LSZ lépésszámláló, majd értékének 0-ra való ellenőrzése következik. Ha az LSZ lépésszámláló tartalma nem 0, az előbbi lépések: balra léptetés, összehasonlítás, összeadás, hányados jegyének beiktatása, az LSZ lépésszámláló tartalmának csökkentése és ellenőrzése addig ismétlődik, amíg az LSZ lépésszámláló tartalma 0 nem lesz. Ekkor az AC(M) részregiszterben levő részmaradék abszolút értékét állítják elő olymódon, hogy mégegyszer 1-es komplemensét képezik. Ezzel az osztási folyamat befejeződött.

A 4.11. és 4.12. folyamatábrákkal leírt osztási eljárás sorrendi műveleteit illusztrálja a 4.13. ábra, amelyen megfigyelhetők a regiszterekben végbemenő lépésenkénti változások. Ebben a példában a következő mennyiségek szerepelnek:

osztandó	$= +00001111$	$= +15_{10}$
osztó	$= -0011$	$= -3_{10}$
hányados	$= -0101$	$= -5_{10}$
maradék	$= +0000$	$= 0$

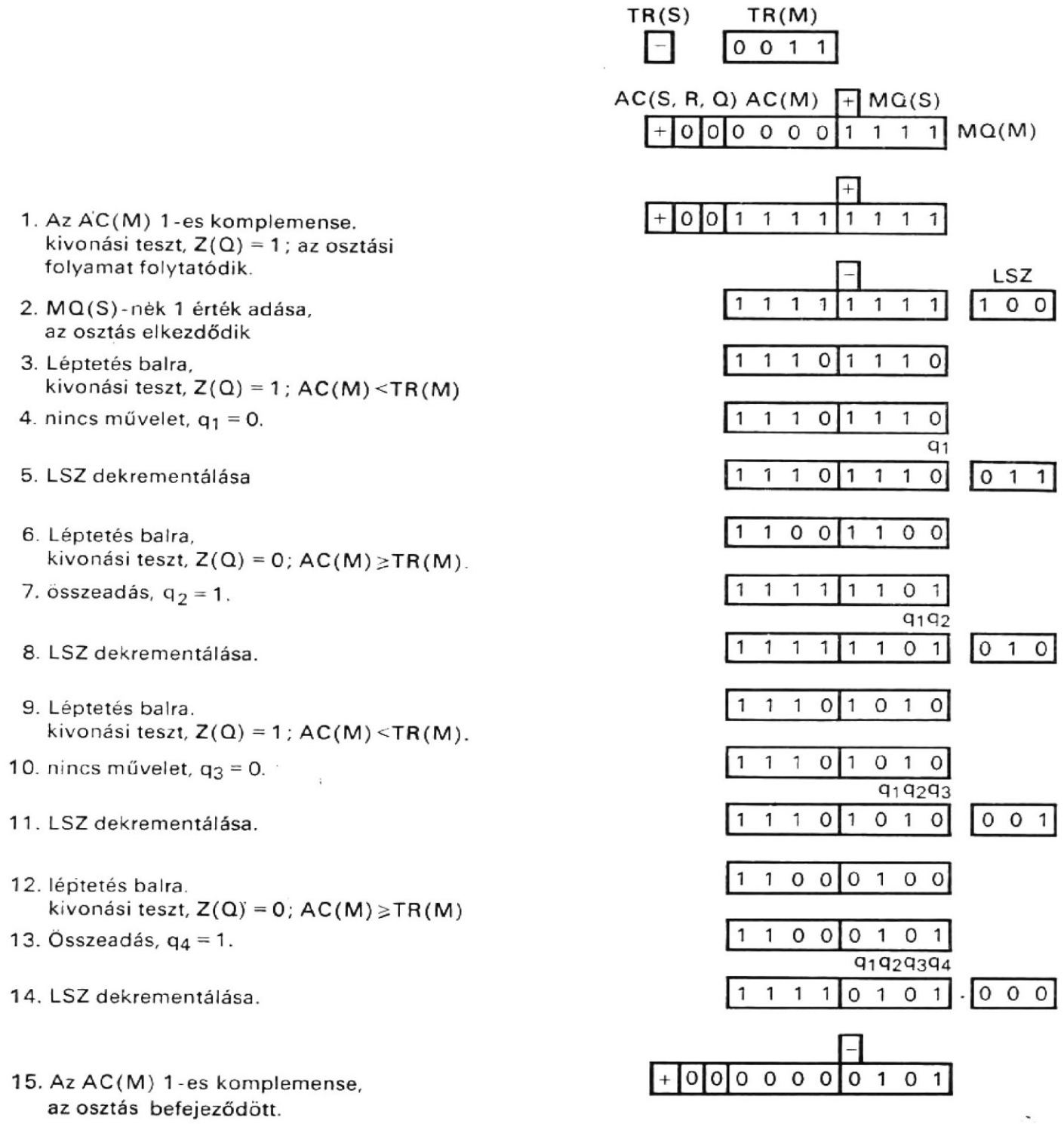
Figyeljük meg, hogy a 4.13. ábrán osztási túlcsordulási vizsgálat (teszt) történt, mielőtt az osztás ténylegesen megkezdődik.

A folyamatábrák alkalmazása mellett az alábbiakban az osztási folyamat procedurális leírását is bemutatjuk.

```
Comment, osztási sorrend                                          (4.9)
/C0/    AC(R,Q,M)←0−0−AC(M)';
        IF (Z(Q)=1) THEN (DVOV←1, GOTO C2);
        IF (TR(S)=AC(S)) THEN (MQ(S)←0) ELSE (MQ(S)←1);
/C/     LSZ←35;
/C1/    AC(M)−MQ(M)←AC(2−35)−MQ(1)'−MQ(2−35)−0;
        IF (Z(Q)=0) THEN (MQ(35)←1, AC(R,Q,M)←0−0−AC(M) add2
        0−0−TR(M)−0;
        LSZ←countdn LSZ;
        IF (LSZ≠0) THEN (GOTO C1);
        AC(M)←AC(M)'
/C2/    END
```

4.3. CDL leírások

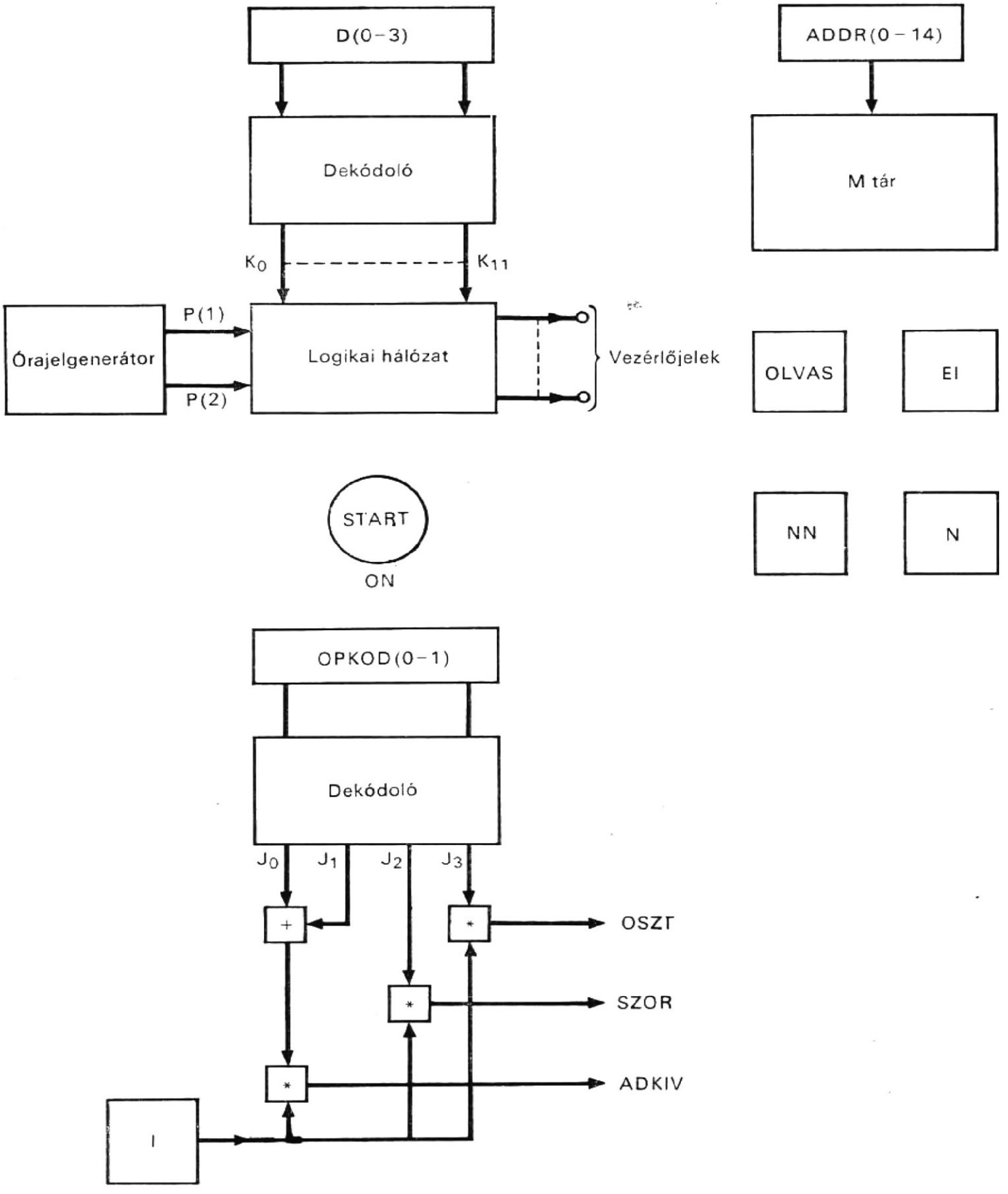
Az összeadási, kivonási, szorzási és osztási algoritmusokat, amelyeket eddig folyamatábrákkal mutattunk be, most CDL utasításokkal írjuk le. Először az időzítő- és vezérlőjeleket írjuk le, ezt követi az összeadási és kivonási utasítássorozat, a szorzási utasítássorozat és végül az osztási utasítássorozat.



4.13. ábra
Előjellel és abszolút értékkel ábrázolt bináris számok fixpontos osztása

4.3.1. Időzítő- és vezérlőjelek

Az összeadási, kivonási, szorzási és osztási folyamatokhoz tartozó struktúra a következő funkcionális elemekből áll: a D 4-bites időzítőregiszterből, az OPKOD 4-bites regiszterből, a hozzájuk tartozó dekódolókból, a VSZ 4-bites várakozásszámlálóból és az ADDR címregiszterrel ellátott M tárból. A tárkapacitás 32 768 36-bites szó. A fentiek mellett alkalmaznak hat darab egybites regisztert, ezek I, OLVAS, EI, N, W és Y. Az I regiszter az utasításlehívó ciklust jelzi, ha 0, és az utasításvégrehajtó ciklust, ha 1. Az



4.14. ábra
Az aritmetikai egység vezérlésének stuktúrája

OLVAS regiszter a tárból való kiolvasási műveletet indítja, ha az értéke 1. Az EI, N, W és Y regiszterek a részfolyamatokhoz kellenek, amint majd szerepüket a későbbiekben bemutatjuk. Végül van egy P órajelgenerátor és egy START kapcsoló.

Az aritmetikai egység működéséhez szükséges struktúra a 4.14. ábrán látható és azt az alábbi CDL utasítások írják le:

Comment, fixpontos aritmetikai egység vezérlésének struktúrája (4.10)

```
Register,    D(0—3),              $időzítőregiszter
             ADDR(0—14)           $a tár címregisztere
             OPKOD(0—3)           $műveletikód-regiszter
             I,                   $ha 0; utasításlehívó ciklus, ha 1: utasítás-
                                  végrehajtó ciklus
             OLVAS,               $parancs a tár kiolvasására
             EI,                  $előjelindikátor
             VSZ(0—3)             $várakozásszámláló
             N,                   $vezérlőregiszter
             W,                   $vezérlőregiszter
             Y,                   $vezérlőregiszter
Memory,      M(ADDR)=M(0—32767, 0—35),
Decoder,     K(0—11)=D,
             J(0—11)=OPKOD,
Switch,      START(ON),
Clock,       P(1—2),
Terminal,    ADKIV=J(0)+J(1)*I,
             SZOR=J(2)*I,
             OSZT=J(3)*I,
/START(ON)/  D←0,
/P(2)/       D←countup D,
```

A fenti struktúra vezérlőjelek sorozatát állítja elő 12 lépésben. A sorozat előállítására a D számlálót, a hozzátartozó dekódolót és a P kétfázisú órajelgenerátort használja fel. Ezt a 12 vezérlőjelet a K(0)*P(1), K(1)*P(1),..., K(11)*P(1) cimkék reprezentálják. Ennek a 12 lépésnek az elvégzését *végrehajtási ciklusnak* nevezzük a továbbiakban.

A sorozat akkor kezdődik, amikor a START kapcsolót ON állásba fordítják; ezzel a START kapcsoló a D regiszternek 0 értéket ad és előállítja az első lépéshez tartozó K(0) vezérlőjelet. A 12 vezérlőjel mindegyike az első órajelfázisban, P(1)-ben keletkezik, míg a D számláló a második, P(2) fázisban inkrementálódik 1-gyel; ily módon halad előre a vezérlőjelek sorozata. Amikor a D számláló eléri a 11-et, a D regisztert 15-re állítja a P(1) órajelfázis, a 12 lépés közül a legutolsó alatt, és ezzel egy 12 lépéses sorozat befejeződött. A D regiszternek 15 értékre állítása egy újabb 12-lépéses vezérlőjel generálását indítja el. A D-t 15 értékre állító mikroművelet kivételével (amelyet az egyes sorozatok végén kell biztosítani) a D számláló vezérléséhez szükséges mikroműveleteket a végrehajtási utasítások írják le. A parancsokat az összeadási és kivonási művelet sorozathoz (ADKIV), a szorzási sorozathoz (SZOR) és az osztási sorozathoz (OSZT) az OPKOD regiszter, a hozzá tartozó dekódoló és az I regiszter állítja elő.

Az aritmetikai folyamatok itt következő leírásaiban egy végrehajtási ciklus 12-lépéses működéséhez szükséges időt úgy választják meg, hogy egyenlő legyen a tár ciklusidejével és így a struktúrában csak egy ciklusidővel kell foglalkozni. Ezen felül minden végrehajtási ciklus kezdete egybeesik mindegyik tárciklus kezdetével. Ez a választás a tárat ciklusról ciklusra működésben tartja, akárcsak az aritmetikai egység is, egyik ciklussal a másik után működésben marad.

Tételezzük fel, hogy a tárból való információ kiolvasását az első lépésben írjuk elő, és a tárból jövő szó a TR regiszterben a negyedik lépés végén áll rendelkezésre.

4.3.2.
Összeadási és kivonási sorozat

A fixpontos összeadási és kivonási folyamatot már leírtuk a 4.6. és 4.7. folyamatábrákkal. A folyamatnak CDL utasításokkal való leírását úgy állíthatjuk elő, hogy vezérlőjelet rendelünk az összeadási és kivonási folyamat minden lépéséhez. Az első hat lépés vezérlőjelei ADKIV * K(0) * P(1),...,ADKIV * K(5) * P(1). Az utolsó lépésnél befejeződik a kezdetiérték-adó rész és a folyamat egy összeadási vagy egy kivonási részre ágazik el, amint a 4.6. és 4.7. ábrán látható. A folyamat további hat lépése hasonlóképpen két 6-lépéses alsorozatra ágaztatható el, az EI regiszter segítségével. A vezérlőjelek a következők: EI' * ADKIV * K(6) * P(1),...,EI' * ADKIV * K(11) * P(1) az összeadási részhez és EI * ADKIV * K(6) * P(1),...,EI * ADKIV * K(11) * P(1) a kivonási részhez. Ezekkel a vezérlőjelekkel a CDL utasításokkal leírt összeadási és kivonási folyamat a következő lesz:

```
Comment, összeadási és kivonási műveletsorozat                    (4.11
/ADKIV * K(0) * P(1)/   OLVAS←1,
/ADKIV * K(1) * P(1)/   N←0, W←0, Y←0, VSZ←0,
/ADKIV * K(2) * P(1)/   AC(R,Q)←0,
/ADKIV * K(3) * P(1)/   TR←M(ADDR),
/ADKIV * K(4) * P(1)/   IF (OPKOD=1) THEN (TR(S)←TR(S)'),
/ADKIV * K(5) * P(1)/   IF (TR(S)=AC(S)), THEN (TR←0) ELSE (EI←1),

Comment, összeadási rész

/EI' * ADKIV * K(7) * P(1)/  AC(R,Q,M)←0—AC(Q,M) add2 0—0—TR(M)—0,
/EI' * ADKIV * K(8) * P(1)/  IF (AC(Q,M)=0) THEN (AC(S)←0),
/EI' * ADKIV * K(9) * P(1)/  IF (AC(Q)=1) THEN (ADOV←1),
/ADKIV * K(11) * P(1)/       I←0, D←15,

Comment, kivonási rész

/EI * ADKIV * K(6) * P(1)/   AC(M)←AC(M)',
/EI * ADKIV * K(7) * P(1)/   AC(R,Q,M)←0—AC(Q,M) add2 0—0—TR(M)—0,
/EI * ADKIV * K(8) * P(1)/   IF (AC(Q)=0 THEN (IF (AC(M)=37...7₈)
                               THEN (AC←0) ELSE (AC(M)←AC(M)')),
/EI * ADKIV * K(9) * P(1)/   IF (AC(Q)=1) THEN (AC)R,Q,M)←1 add2
                               0—AC(Q,M)—0, AC(S)←AC(S)'),
/EI * ADKIV * K(10) * P(1)/  AC(Q)←0,
                             END
```

A fenti leírásban a tárból való olvasás művelete az első lépésben kap parancsot, és a szó a tár a negyedik lépésben áll rendelkezésre, amint azt már az előzőkben feltételeztük. Az utolsó lépés során a D regiszter 15 értéket kap, annak biztosítására, hogy egy másik ciklus kezdődjék, és ez a ciklus a tárciklus kezdetével esik egybe. Ezenfelül az I regiszter 0 értéket kap, hogy a következő ciklus utasításlehívó ciklus legyen, és így a soron következő utasítást hívja le a tárból.

4.3.3.
Szorzási sorozat

A fixpontos szorzási folyamatot a 4.8. és 4.9. folyamatábrán írtuk le. A folyamat előírt sorrendjéhez szükséges vezérlőjelek SZOR * K(0) * P(1),...,SZOR * K(11) * P(1). Ismét feltételezzük, hogy a tárból kiolvasott szó a negyedik lépés végén áll rendelkezésre. A CDL utasításokkal leírt szorzási folyamat ezekkel a vezérlőjelekkel a következő lesz:

```
Comment, szorzási folyamat                                        (4.12)
/SZOR * K(0) * P(1)/      OLVAS←1,
/SZOR * K(1) * P(1)/      N←0, W←0, Y←0, VSZ←0,
/SZOR * K(2) * P(1)/      AC(R,Q,M)←0,
/SZOR * K(3) * P(1)/      TR←M(ADDR),
/SZOR * K(4) * P(1)/      IF (TR(S)=MQ(S)) THEN (AC(S)←0)
                                   ELSE (AC(S)←1),
/SZOR * K(5) * P(1)/      IF ((TR(M)=0+(MQ(M)=0))
                            THEN (MQ←0, AC(S)←0, Y←1) ELSE (W←1),
/SZOR * K(11) * P(1)/     IF (Y=1) THEN (Y←0, I←0, D←15),
                          IF(W=1) THEN (N←1, W←0, D←5),

Comment, szorzási rész
/N * SZOR * K(6) * P(1)/  LSZ←18, C←0, MQ(S)←0,
/N * SZOR * K(7) * P(1)/  IF ((C=1)*(MQ(34,35)=00)
                            THEN (MQ(34,35)←01, C←0),
                          IF ((C=1)*(MQ(34,35)=01)
                            THEN (MQ(34,35)←10, C←0),
                          IF ((C=1)*(MQ(34,35)=10) THEN (MQ(34,35)←11),
                          IF ((C=1)*(MQ(34,35)=11) THEN (MQ(34,35)←00),
/N * SZOR * K(8) * P(1)/  IF (MQ(34,35)=01) THEN (AC(R,Q,M)
                            ←AC(R,Q,M) add2 0—0—TR(M)—0),
                          IF (MQ(34,35)=10) THEN (AC(R(R,Q,M)
                            ←AC(R,Q,M) add2 0—TR(M)—0—0),
                          IF (MQ(34,35)=11) THEN (AC(R,Q,M)
                            ←0—0—AC(M) add2 1—1—TR(M)'—1,C←1),
/N * SZOR * K(9) * P(1)/  AC(2—35)—MQ←AC(Q,M)—MQ(S,1—33),
                            AC(Q,1)←AC(R)—AC(R), LSZ←countdn LSZ,
/N * SZOR * K(10) * P(1)/ IF (LSZ=0) THEN (MQ(S)←AC(S),AC(M)←AC(2—35)
                            —MQ(S)) ELSE (D←6), VSZ←4,
/N * SZOR * K(11) * P(1)/ IF (LSZ=0) THEN (N←0, I←0, D←15)
                                   ELSE (D←10, VSZ←countdn VSZ),
                          END
```

Az előbbi leírás két részből áll: a 4.8. folyamatábrával leírt kezdetiérték-adásból és a 4.9. folyamatábrán látható szorzásból. Ha akár a szorzó, akár a szorzandó nulla, a szorzat nulla lesz. Az Y regiszter 1 értéket kap. Ekkor a fixpontos szorzási folyamat lezárul az első végrehajtási ciklus utolsó lépésekor. Ha sem a szorzandó, sem a szorzó nem nulla, az N regiszter 1 értéket kap, és ez indítja a szorzási részsorozatot.

A szorzási részben van egy hurok, amely négy lépésből áll. A szorzási hurokban a következő hat vezérlőjel látja el a sorrendi vezérlést: N * SZOR * K(6) * P(1),...,N * * SZOR * K(11) * P(1). Az egyes iterációk alatt az LSZ lépésszámláló értékét nullára ellenőrzik. Ha nem nulla, a D számláló 6 értéket kap, és így mégegyszer generálódik a vezérlőjelek sorozata. A szorzási hurokban 18 iteráció van. Amikor a lépésszámláló

nulla lesz, akkor a végrehajtási ciklus hetedik lépésében vagyunk. Ebben a lépésben a VSZ számláló 4 értéket kap. Amint a fenti, (4.12) leírás utolsó utasításában megadtuk, a VSZ számláló ezután az egyutasításos hurokban addig dekrementálódik, amíg értéke eléri a 0 értéket. Ekkor a sorozat pontosan a végrehajtási ciklus utolsó lépésénél tart. Mind az N, mind az I regiszter 0 értéket kap, míg a D regiszter 15 lesz, hogy indíthasson egy másik ciklust.

4.3.4. Osztási sorozat

A fixpontos osztási folyamatot a 4.11. és 4.12. folyamatábrákkal írtuk le. A kezdetiérték-adó rész előírt sorrendjének biztosításához szükséges vezérlőjelek a következők: OSZT * K(0) * P(1),...,OSZT * K(4) * P(1) és OSZT * K(11) * P(1). Az osztási részt időzítő- és vezérlőjelek: N * OSZT * K(8) * P(1),...,N * OSZT * K(11) * P(1). A tárból kiolvasott szó ismét csak az első végrehajtási ciklus negyedik lépésének a végén áll rendelkezésre. A fixpontos osztást a CDL utasítások ezekkel a vezérlőjelekkel a következőképpen írják le:

```
Comment, osztási sorozat                                        (4.13)
/OSZT * K(0) * P(1)/      OLVAS ← 1,
/OSZT * K(1) * P(1)/      W ← 0, N ← 0, Y ← 0, VSZ ← 0,
/OSZT * K(2) * P(1)/      AC(R,Q,M) ← 0 — 0 — AC(M)',
/OSZT * K(3) * P(1)/      TR ← M(ADDR),
/OSZT * K(4) * P(1)/      IF (Z(Q)=0) THEN (DVOV ← 1, W ← 1)
                            ELSE (LSZ ← 35,N ← 1),
                          IF (TR(S)=AC(S)) THEN (MQ(S) ← 0)
                            ELSE (MQ(S) ← 1),
/OSZT * K(11) * P(1)/     IF (W=1) THEN (W ← 0, I ← 0, D ← 15),

Comment, osztási rész
/N * OSZT * K(8) * P(1)/  AC(M) — MQ(M) ← AC(2 — 35) — MQ(1)' — MQ(2 — 35) — 0,
                            LSZ ← countdn LSZ,
/N * OSZT * K(9) * P(1)/  IF (Z(Q)=1) THEN (AC(R,Q,M) ← 0 — 0 — AC(M) add2
                            0 — 0 — TR(M) — 0, MQ(35) ← 1),
/N * OSZT * K(10) * P(1)/ IF (LSZ≠0) THEN (D ← 7)
                            ELSE (AC(M) ← AC(M)', VSZ ← 6),
/N * OSZT * K(11) * P(1)/ IF (VSZ=0) THEN (N ← 0, I ← 0, D ← 15)
                            ELSE (D ← 10,VSZ ← countdn VSZ),
                          END
```

A fenti leírásban, ha a kezdetiérték-adó részben osztási túlcsordulás következik be, a W regiszter 1 értéket kap és a fixpontos osztás befejeződik az első végrehajtási ciklus végén. Egyébként az N regiszter 1 értéket kap, és ez indítja az osztási részsorozatot. Az osztási részsorozatban található egy háromlépéses osztási hurok. A hurok minden egyes iterációjakor nulla értékre ellenőrzik a LSZ lépésszámlálót. Ha az nem nulla, a D számláló 7 értéket kap és így ismétlődik a részsorozat vezérlőjeleinek sorozata. Az osztási hurokban 35 interáció van. Amikor a lépésszámláló nulla lesz, akkor a végrehajtási ciklus ötödik lépésében vagyunk, és a VSZ számláló 6 értéket kap. Ezután a VSZ számláló az egyutasításos hurokban [az utolsó utasítás a (4.14)-ben] addig dekrementálódik, amíg el nem éri a nullát. Ekkor a sorozat ennek a végrehajtási ciklusnak az utolsólépéséhez jutott el, és ugyanúgy fejeződik be, mint a szorzási sorozat.

4.4. Párhuzamos összeadó csoport- és szekcióátvitellel

A párhuzamos összeadót a (4.3) és (4.4) terminál- (kapocspár-) utasítások írták le, jelképi jelölése a 4.4. és 4.5. ábrákon látható. Minthogy a párhuzamos összeadó összeadási ideje döntően befolyásolja az aritmetikai egység számítási sebességét, ez a szakasz mintegy illusztrációképpen egy másik típusú párhuzamos összeadót mutat be, amely csoportátviteleket és szekcióátviteleket használ fel annak érdekében, hogy az összeadási idő csökkenjen.

A következőkben a *szint* fogalmat használjuk egy logikai ÉS, ill. egy logikai VAGY áramkör leírására. Például ha a jel tíz ilyen, sorbakötött logikai áramkörön halad át, akkor azt mondjuk, hogy a jel tíz szinten ment keresztül. Minél több szinten megy keresztül a jel, annál több időre van szüksége a terjedéshez. Ezért, a szinteknek azt a számát, amelyen keresztül kell haladnia egy átvitel- vagy összegbitnek, egyben az idő mértékének fogjuk itt tekinteni.

4.4.1. Egybites teljes összeadó

A teljes összeadóhoz tartozó Boole-függvényeket már leírtuk a (4.1) és (4.2) összefüggésekben. E két egyenlet helyett a teljesösszeadó más módon is leírható, ahogy azt az alábbiakban bemutatjuk.

Legyen $P(i)=ADTR(i)+ADAC(i)$ (4.14)

és $CX(i)=ADTR(i)\odot ADAC(i)$ (4.15)

vagy $CX(i)=ADTR(i)'*ADAC(i)'+ADTR(i)*ADAC(i)$ (4.16)

P(i) 1 lesz, ha vagy az ADAC(i) összeadandó- vagy az ADTR(i) összeadandóbit vagy mindkettő 1. Tehát P(i) azt a feltételt reprezentálja, hogy az i-edik teljes összeadó az i-edik helyen keletkezett átvitelt továbbítja, ha van ilyen; ezt nevezik *átvitelterjedésnek*. Így $P(i)*C(i)$ reprezentálja azt a C(i) bemeneti átvitelt, amely az i-edik szakaszon terjedt tovább. CX(i) 1 lesz, ha a két összeadandóbit egyaránt 0, vagy egyaránt 1. Ekkor $P(i)*CX(i)$ reprezentálja a generált átvitelt, minthogy $P(i)*CX(i)$ csak akkor 1, ha a két összeadandóbit egyaránt 1. Mivel a $C(i-1)$ kimeneti átvitel vagy generált átvitel, vagy továbbított átvitel, ezért

$$C(i-1)=P(i)*(CX(i)+C(i)).$$

A teljes összeadót most már a következő két Boole-függvénnyel lehet leírni:

$ADD(i)=CX(i)\odot C(i)$ (4.17)

$C(i-1)=P(i)*(CX(i)+C(i))$ (4.18)

Vegyük észre, hogy a (4.17)-ben leírt összegbit azonos a (4.5)-ben leírttal, ha a (4.17)-ben szereplő CX(i) mennyiséget a (4.5)-ben szereplővel helyettesítjük.

4.4.2.
A párhuzamos összeadó szervezése

A párhuzamos összeadó a (4.17) és (4.18) Boole-függvényekkel leírt egybites teljes összeadók felhasználásával építhető fel. Az így felépített 37-bites párhuzamos összeadót tömören leírhatjuk az alábbi terminál- (kapocspár-) utasítással:

Terminal,
$$P(Q,1-35)=ADTR(Q,1-35)+ADAC(Q,1-35), \quad (4.19)$$
$$CX(R,Q,1-35)=ADTR(R,Q,1-35)\odot ADAC(R,Q,1-35), \quad (4.20)$$
$$ADD(R,Q,1-35)=CX(R,Q,1-35)\odot C(R,Q,1-35), \quad (4.21)$$
$$C(R,Q,1-34)=P(Q,1-35)*(CX(Q,1-35)+C(Q,1-35)), \quad (4.22)$$

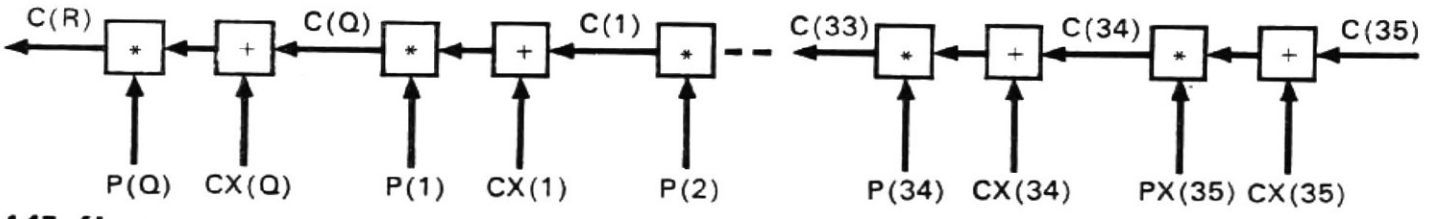
A (4.22) összefüggés által leírt átviteleket *helyi átviteleknek* nevezzük. A párhuzamos összeadó helyi átviteleit ábrázoló logikai hatásvázlatot lehet szerkeszteni a fenti terminálutasításból; ezt a 4.15. ábra mutatja, amelyen látható az az út, amelynek mentén terjed az átvitel. Ezt az utat az *átvitel terjedési útjának* nevezzük, az így felépített párhuzamos összeadót pedig kapuzott átvitelű összeadónak. Ahhoz, hogy egy helyi átvitel továbbterjedjen az átviteli út mentén, szakaszonként két szinten kell áthaladnia. Így,

4.1. táblázat

Bemeneticsoport- és szekcióátvitelek és szintjeik

Helyi átvitel		Bemeneticsoport-átvitel		Bemeneticsoport- vagy szekcióátvitel		Összegbit	
név	szint	név	szint	név	szint	név	szint
C(35)	0	IG(1)	0	IS(1)	0	ADD(35)	2
C(31)	8	IG(2)	5	IG(2)	5	ADD(31)	7
C(27)	16	IG(3)	7	IG(3)	7	ADD(27)	9
C(23)	24	IG(4)	9	IS(2)	7	ADD(23)	9
C(19)	32	IG(5)	11	IG(5)	9	ADD(19)	11
C(15)	40	IG(6)	13	IG(6)	11	ADD(15)	13
C(11)	48	IG(7)	15	IS(3)	9	ADD(11)	11
C(7)	56	IG(8)	17	IG(8)	11	ADD(7)	13
C(3)	64	IG(9)	19	IG(9)	13	ADD(3)	15
C(R)	72	IG(10)	21	IS(4)	11	ADD(R)	13

C(35), C(34), C(33),..., C(1), C(Q) és C(R) helyi átvitelek fordulhatnak elő, rendre, 0, 2, 4,..., 68, 70 és 72 szinten áthaladva. Tíz olyan helyi átvitel van: a C(35),..., C(3) és C(R), amelyek különleges érdeklődésre tarthatnak számot. Ezt a tíz helyi átvitelt és azokat a szinteket, amelyeken áthaladhatnak, a 4.1. táblázat első és második oszlopa mutatja. Hasonlóképpen kimutatható, hogy az ADD(35), ADD(34), ADD(33),..., ADD(2) és ADD(1) összeg kapocspárokon a jelek rendre, 2, 4, 6,..., 78, 70 szinten való áthaladás után fordulnak elő.



4.15. ábra
A párhuzamos összeadó átvitelterjedésének szemléltetése

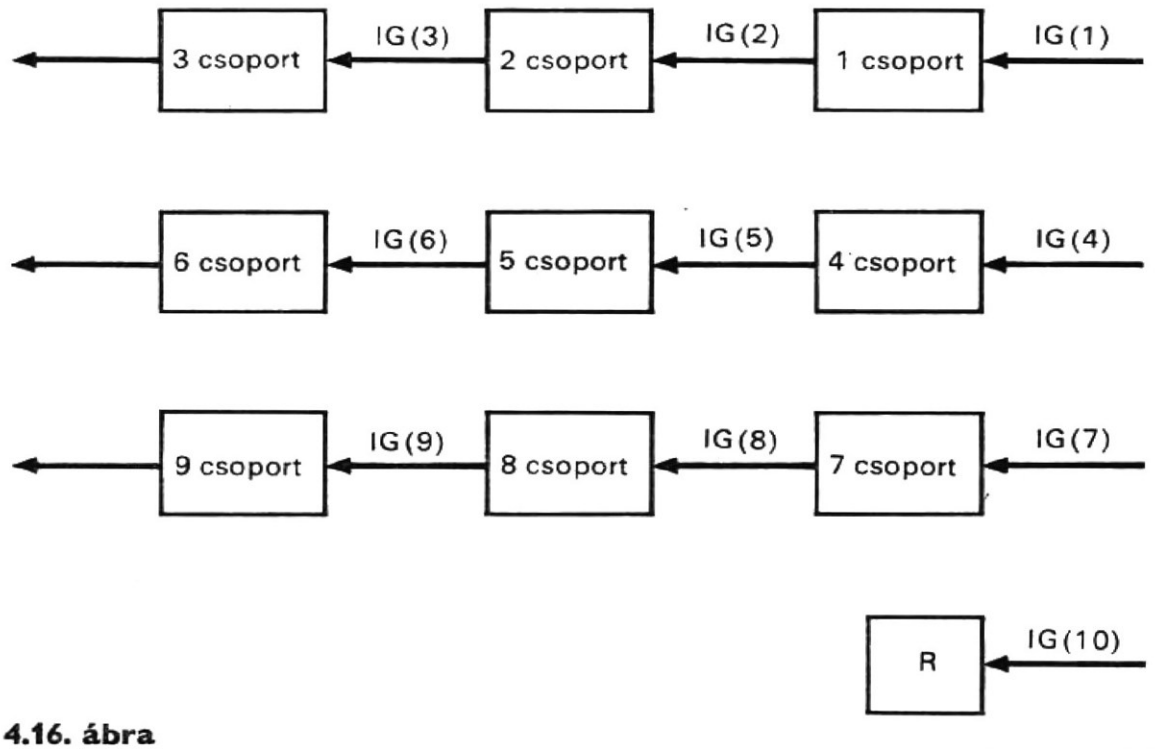
A fenti párhuzamos összeadóban az átvitel terjedése lényegesen több időt igényel, mint az kívánatos lenne, mivel minél hosszabb a terjedési idő, annál hosszabb az összeadási idő. Minthogy a szorzás ismételt összeadásokkal, az osztás ismételt kivonásokkal, és a kivonás komplemensek összeadásával történik, az átvitelterjedési idő csökkenése a maga részéről csökkenti az összeadás, kivonás, szorzás és osztás idejét. Ez az oka annak, hogy a párhuzamos összeadóban az átvitelterjedési idő csökkentésének nagy jelentősége van az aritmetikai egység sebessége szempontjából.

Az egyik közelítés az átvitelterjedési idő csökkentésére az, hogy a teljesösszeadó szakaszokat *csoportokba* szervezzük. Kombináljuk csoportba a teljesösszeadók négy-négy szakaszát. Így, a 35-ik, 34-ik, 33-ik és 32-ik szakasz kerül az 1. csoportba; a 31-ik, 30-ik, 29-ik és 28-ik a 2. csoportba és így tovább. A 37 teljesösszeadó szakasz 10 csoportból áll, de a 10-ik csoport csak egy szakaszt tartalmaz. Továbbá szervezzünk minden három csoportból egy *szekciót*. Így, az 1., 2. és 3. csoport szervezéséből alakul az 1. szekció; a 4., 5. és 6. csoportból a 2. szekció és így tovább. A tíz csoportból 4 szekció képződik, de a 4-ik szekció mindössze egy teljesösszeadó szakaszból áll.

Minden csoporthoz külön logikai hálózat tartozik, a *bemeneticsoport-átvitelek*, az IG(1), IG(2),..., IG(10) generálására. A 4.16. ábra blokkdiagramja mutatja a négy szekcióba osztott tíz csoportból álló logikai hálózatot. Ez a tíz bemeneticsoport-átvitel helyettesíti rendre a C(35), C(31),..., C(Q) és C(R) helyi átvitelt, az átvitelterjedési idő csökkentése céljából. A tíz bemeneticsoport-átvitelt és azokat a szinteket, amelyeken áthaladhatnak, miután a tíz helyi átvitelt a bemeneticsoport-átvitelekkel helyettesítettük, a 4.1. táblázat harmadik és negyedik oszlopa mutatja. Figyeljük meg, hogy a szintek száma, amelyeken a helyi átvitelek áthaladhatnak, nagymértékben lecsökkent. Például, a C(R) átvitelhez szükséges szintszám 72-ről 21-re csökkent.

Ezeket a bemeneticsoport-átviteleket leíró Boole-egyenleteket később fogjuk levezetni. A bemeneticsoport-átviteleket definiáló terminálutasítást az alábbiakban mutatjuk be.

Terminal,
$$IG(1)=C(35) \tag{4.23}$$
$$IG(2)=G(1)+IG(1)*PG(1) \tag{4.24}$$
$$IG(3)=G(2)+IG(2)*PG(2) \tag{4.25}$$
$$IG(4)=G(3)+IG(3)*PG(3) \tag{4.26}$$



4.16. ábra
Logikai hálózat a három szekció kilenc csoportjában a csoportátvitelek generálására

$$IG(5)=G(4)+IG(4)*PG(4) \quad (4.27)$$
$$IG(6)=G(5)+IG(5)*PG(5) \quad (4.28)$$
$$IG(7)=G(6)+IG(6)*PG(6) \quad (4.29)$$
$$IG(8)=G(7)+IG(7)*PG(7) \quad (4.30)$$
$$IG(9)=G(8)+IG(8)*PG(8) \quad (4.31)$$
$$IG(10)=G(9)+IG(9)*PG(9) \quad (4.32)$$

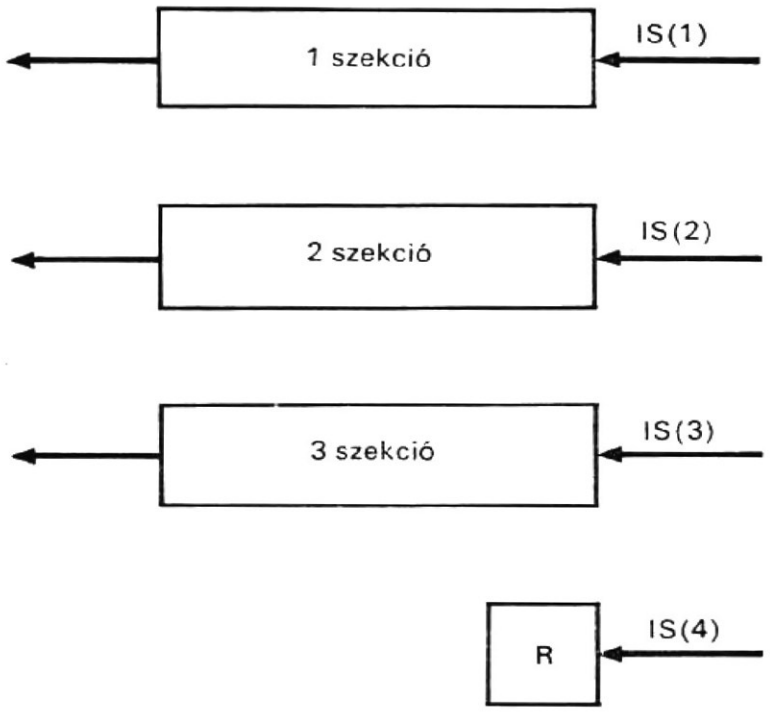
ahol G(1), G(2),..., G(9) elnevezése *csoportátvitel* és a PG(1), PG(2),..., PG(9) *csoportterjedés*. Mindezeket a 4.16. ábrán látható logikai hálózat generálja.

Egy-egy szekcióhoz is hozzá van rendelve egy külön logikai hálózat, az IS(1),...,IS(4) *bemenetiszekció-átvitelek* generálására. A 4.17. ábrán látható blokkdiagram ábrázolja három szekcióhoz tartozó logikai hálózatokat. Ahhoz, hogy tovább csökkentsük az átvitel terjedési időt, ezekkel a bemenetiszekció-átvitelekkel helyettesítjük a négy, IG(1), IG(4), IG(7) és IG(10) bemeneticsoport-átviteleket. A hat bemeneticsoport-átvitelt, a négy bemenetiszekció-átvitelt és azoknak a szinteknek a számát, amelyeken keresztül haladhatnak, a 4.1. táblázat ötödik és hatodik oszlopa mutatja. Amint a táblázatból látható, azoknak a szinteknek a száma, amelyeken ezek az átvitelek továbbterjedhetnek, még tovább csökkent. A szintek maximális száma a negyedik oszlopban levő 21-ről a hatodik oszlopban szereplő 13-ra csökkent. Egyetlen átvitelnek sem kell 13-nál több szinten keresztülhaladnia, és amint a 4.1. táblázat hetedik és nyolcadik oszlopából látható, egyetlen összegbitnek sem kell 15-nél több szinten tovaterjednie.

A bemenetiszekció-átvitelek leírására szolgáló Boole-egyenleteket később vezetjük le. A bemenetiszekció-átviteleket definiáló terminálutasítást az alábbiakban mutatjuk be:

Terminal,
$$IS(1)=C(35) \quad (4.33)$$
$$IS(2)=S(1)+IS(1)*PS(1) \quad (4.34)$$
$$IS(3)=S(2)+IS(2)*PS(2) \quad (4.35)$$
$$IS(4)=S(3)+IS(3)*PS(3) \quad (4.36)$$

ahol S(1), S(2) és S(3) a *szekcióátvitel* és a PS(1), PS(2) és PS(3) a *szekcióterjedés*. Valamennyit közülük a 4.17. ábra logikai hálózatai generálják.



4.17. ábra
Logikai hálózat a három szekcióban a szekcióátvitelek generálására

4.4.3.
Terminálutasítások a párhuzamos összeadók leírására

A mind csoport-, mind szekcióátvitellel rendelkező párhuzamos összeadót a következő terminál- (kapocspár-) utasítások definiálják:

1. A teljes összeadók terminalutasításai, ezeket a (4.19)...(4.22) Boole-függvények tartalmazzák, kivéve a C(35), C(31), C(27), C(23), C(19), C(15), C(11), C(7), C(3) és C(R) helyi átviteleket, mert azokat sorrendben az IG(1), ..., IG(10) bemeneti-csoport-átvitelek helyettesítik.
2. A csoportátvitelek terminálutasításai, amelyeket a (4,45), (4.47)...(4.54) Boole-függvények írnak le, és a csoportterjedések terminálutasításai, amelyeket a (4.55) és (4.56) függvények adnak meg.
3. A bemeneticsoport-átvitelek terminálutasítása, amelyeket a (4.23)...(4.32) Boole-függvények írnak le, kivéve az IG(1), IG(4) és IG(10) bemeneticsoport-átviteleket, amelyeket az IS(1), ..., IS(4) bemenetiszekció-átvitelek helyettesítenek.
4. A szekcióátvitelek terminálutasítása, amelyeket a (4.57)...(4.59) Boole-függvények írnak le, a szekcióterjedések terminálutasításai, a (4.60)...(4.62) függvényekkel megadva, valamint a bemenetiszekció-átviteleké, a (4.33)...(4.36) függvényekkel leírva.

Ezeket a terminál- (kapocspár-) utasításokat az alábbiakban mutatjuk be.

Comment, a párhuzamos összeadó bemenetei (4.37)

Terminal, ADAC(R,Q,1—35),
ADTR(R,Q,1—35),
C(35),

Comment, összegterminálok (kapocspárok)

Terminal, ADD(R)=CX(R)⊙IS(4),
ADD(Q,1—2)=CX(Q,1—2)⊙C(Q,1—2),
ADD(3)=CX(3)⊙IG(9),
ADD(4—6)=CX(4—6)⊙C(4—6)
ADD(7)=CX(7)⊙IG(8),
ADD(8—10)=CX(18—10)⊙C(8—10),
ADD(11)=CX(11)⊙IS(3),
ADD(12—14)=CX(12—14)⊙C(12—14),
ADD(15)=CX(15)⊙IG(6),
ADD(16—18)=CX(16—18)⊙C(16—18),
ADD(19)=CX(19)⊙IG(5),
ADD(20—22)=CX(20—22)⊙C(20—22),
ADD(23)=CX(23)⊙IS(2),
ADD(24—26)=CX(24—26)⊙C(24—26),
ADD(27)=CX(27)⊙IG(3),
ADD(28—30)=CX(28—30)⊙C(28—30),
ADD(31)=CX(31)⊙IG(2),
ADD(32—34)=CX(32—34)⊙C(32—34),
ADD(35)=CX(35)⊙IS(1),

Comment, helyi átvitelek és átvitelterjedések

Terminal, C(Q, 1—2)=P(1—3)*(CX(1—3)+C(1—3))
C(3)=IG(9),

C(4−6)=P(5−7)∗(CX(5−7)+C(5−7)),
C(7)=IG(8),
C(8−10)=P(9−11)∗(CX(9−11)+C(9−11)),
C(11)=IS(3),
C(12−14)=P(13−15)∗(CX(13−15)+C(13−15)),
C(15)=IG(6),
C(16−18)=P(17−19)∗(CX(17−19)+C(17−19)),
C(19)=IG(5),
C(20−22)=P(21−23)∗(CX(21−23)+C(21−23)),
C(23)=IS(2),
C(24−26)=P(25−27)∗(CX(25−27)+C(25−27)),
C(27)=IG(3),
C(28−30)=P(29−31)∗(CX(29−31)+C(29−31)),
C(31)=IG(2),
C(32−34)=P(33−35)∗(CX(33−35)+C(33−35)),
P(Q,1−35)=ATTR(Q,1−35)+ADAC(Q,1−35),
CX(R,Q,1−35)=ATTR(R,Q,1−35)⊙ADAC(R,Q,1−35),

Comment, bemeneticsoport-átvitelek és bemenetiszekció-átvitelek

Terminal, Is(1)=C(35),
IG(2)=G(1)+IS(1)∗PG(1),
IG(3)=G(2)+IG(2)∗PG(2),
IS(2)=S(1)+IS(1)∗PS(1),
IG(5)=G(4)+IS(2)∗PG(4),
IG(6)=G(5)+IG(5)∗PG(5),
IS(3)=S(2)+IS(2)∗PS(2),
IG(8)=G(7)+IS(3)∗PG(7),
IG(9)=G(8)+IG(8)∗PG(8),
IS(4)=S(3)+IS(3)∗PS(3),

Comment, csoportterjedések

Terminal, PG(1)=P(32)∗P(33)∗P(34)∗P(35),
PG(2)=P(28)∗P(29)∗P(30)∗P(31),
PG(3)=P(24)∗P(25)∗P(26)∗P(27),
PG(4)=P(20)∗P(21)∗P(22)∗P(23),
PG(5)=P(16)∗P(17)∗P(18)∗P(19),
PG(6)=P(12)∗P(13)∗P(14)∗P(15),
PG(7)=P(8)∗P(9)∗P(10)∗P(11),
PG(8)=P(4)∗P(5)∗P(6)∗P(7),
PG(9)=P(Q)∗P(1)∗P(2)∗P(3),

Comment, szekcióterjedések

Terminal, PS(1)=PG(1)∗PG(2)∗PG(3),
PS(2)=PG(4)∗PG(5)∗PG(6),
PS(3)=PG(7)∗PG(8)∗PG(9),

Comment, szekcióátvitelek

Terminal, S(1)=G(3)+G(2)∗PG(2)∗PG(3)+G(1)∗
∗PG(1)∗PG(2)∗PG(3),
S(2)=G(6)+G(5)∗PG(5)∗PG(6)+
+G(4)∗PG(4)∗PG(5)∗PG(6),
S(3)=G(9)+G(8)∗PG(8)∗PG(9)+
+G(7)∗PG(7)∗PG(8)∗PG(9),

```
Comment, csoportátvitelek
Terminal, G(1)=P(32)*(CX(32)+P(33))
                *(CX(32)+CX(33)+P(34))
                *(CX(32)+CX(33)+CX(34)+P(35))
                *(CX(32)+CX(33)+CX(34)+CX(35)),
G(2)=P(28)*(CX(28)*P(29))
                *(CX(28)+CX(29)+P(30))
                *(CX(28)+CX(29)+CX(30)+P(31))
                *(CX(28)+CX(29)+CX(30)+CX(31)),
G(3)=P(24)*(CX(24)+P(25))
                *(CX(24)+CX(25)+P(26))
                *(CX(24)+CX(25)+CX(26)+P(27))
                *(CX(24)+CX(25)+CX(26)+CX(27)),
G(4)=P(20)*(CX(20)+P(21))
                *(CX(20)+CX(21)+P(22))
                *(CX(20)+CX(21)+CX(22)+P(23))
                *(CX(20)+CX(21)+CX(22)+CX(23)),
G(5)=P(16)*(CX(16)+P(17))
                *(CX(16)+CX(17)+P(18))
                *(CX(16)+CX(17)+CX(18)+P(19))
                *(CX(16)+CX(17)+CX(18)+CX(19)),
G(6)=P(12)*(CX(12)+P(13))
                *(CX(12)+CX(13)+P(14))
                *(CX(12)+CX(13)+CX(14)+P(15))
                *(CX(12)+CX(13)+CX(14)+CX(15)),
G(7)=P(8) *(CX(8)+P(9))
                *(CX(8)+CX(9)+P(10))
                *(CX(8)+CX(9)+CX(10)+P(11))
                *(CX(8)+CX(9)+CX(10)+CX(11)),
G(8)=P(4) *(CX(4)+P(5))
                *(CX(4)+CX(5)+P(6))
                *(CX(4)+CX(5)+CX(6)+P(7))
                *(CX(4)+CX(5)+CX(6)+CX(7)),
G(9)=P(Q) *(CX(Q)+P(1))
                *(CX(Q)+CX(1)+P(2))
                *(CX(Q)+CX(1)+CX(2)+P(3))
                *(CX(Q)+CX(1)+CX(2)+CX(3)),
```

4.4.4. Csoportátvitelek

A következőkben meghatározzuk a G(i) csoportátvitelekhez, a PG(i) csoportterjedésekhez és az IG(i) bemeneticsoport-átvitelekhez tartozó logikai egyenleteket. Az első csoport a teljes összeadók 35-ik, 34-ik, 33-ik és 32-ik szakaszaiból áll. A (4.22)-ből kapjuk, hogy:

```
C(34)=P(35)*(CX(35)+C(35)),                    (4.38)
C(33)=P(34)*(CX(34)+C(34)),                    (4.39)
C(32)=P(33)*(CX(33)+C(33)),                    (4.40)
C(31)=P(32)*(CX(32)+C(32)),                    (4.41)
```

A csoportátviteleket a csoporton belüli helyi átvitelek és bemeneticsoport-átvitel felhasználása nélkül állítjuk elő. Ezért a G(1) csoportátvitelt úgy kaphatjuk meg a (4.38)...(4.41) Boole-függvényekből, hogy belőlük lépésről lépésre eliminálják a C(32)...C(35) helyi átviteleket, továbbá a C(35) bemeneti átvitelt nullának tekintjük. Így a (4.38)-ból a következőket kapjuk:

$$C(34)=P(35)*CX(35) \tag{4.42}$$

A (4.42)-ben szereplő C(34)-et behelyettesítve a (4.39)-be egyszerűsítés után kapjuk, hogy

$$C(33)=P(34)*(CX(34)+P(35)*(CX(34)+CX(35)) \tag{4.43}$$

A (4.43)-ban levő C(33)-at behelyettesítve a (4.40)-be, egyszerűsítés után felírhatjuk, hogy

$$\begin{aligned} C(32)=P(33)&*(CX(33)+P(34)\\ &*(P(35)+CX(33)+CX(34)\\ &*(CX(33)+CX(34)+CX(35)) \end{aligned} \tag{4.44}$$

A (4.44)-ben levő C(32)-őt behelyettesítve a (4.41)-be, az eredményül kapott C(31) átvitel az első csoport G(1) csoportátvitele lesz. Tehát:

$$\begin{aligned} G(1)=P(32)&*(CX(32)+P(33))\\ &*(CX(32)+CX(33)+P(34))\\ &*(CX(32)+CX(33)+CX(34)+P(35))\\ &*(CX(32)+CX(33)+CX(34)+CX(35)) \end{aligned} \tag{4.45}$$

A második csoport a 31-ik, 30-ik, 29-ik és 28-ik szakaszt tartalmazza. A (4.22)-ből kapjuk, hogy

$$\begin{aligned} &C(30)=P(31)*(CX(31)+C(31)),\\ &C(29)=P(30)*(CX(30)+C(30)),\\ &C(28)=P(29)*(CX(29)+C(29)),\\ &C(27)=P(28)*(CX(28)+C(28)), \end{aligned} \tag{4.46}$$

A (4.46) ismét figyelmen kívül hagyja a C(31) bemeneti átvitelt. Ezen függvények behelyettesítése és egyszerűsítése után az eredményül kapott C(27) átvitel lesz a második csoportban a G(2) csoportátvitel. Így a következőket kapjuk:

$$\begin{aligned} G(2)=P(28)&*(CX(28)+P(29))\\ &*(CX(28)+CX(29)+P(30))\\ &*(CX(28)+CX(29)+CX(30)+P(31))\\ &*(CX(28)+CX(29)+CX(30)+CX(31)), \end{aligned} \tag{4.47}$$

A többi csoport átviteleihez tartozó Boole-függvényeket hasonlóan lehet levezetni. Ezek:

$$\begin{aligned} G(3)=P(24)&*(CX(24)+P(25))\\ &*(CX(24)+CX(25)+P(26))\\ &*(CX(24)+CX(25)+CX(26)+P(27))\\ &*(CX(24)+CX(25)+CX(26)+CX(27)), \end{aligned} \tag{4.48}$$

$$\begin{aligned} G(4)=P(20)&*(CX(20)+P(21))\\ &*(CX(20)+CX(21)+P(22))\\ &*(CX(20)+CX(21)+CX(22)+P(23))\\ &*(CX(20)+CX(21)+CX(22)+CX(23)), \end{aligned} \tag{4.49}$$

$$\begin{aligned} G(5)=P(16)&*(CX(16)+P(17))\\ &*(CX(16)+CX(17)+P(18)) \end{aligned}$$

$$*(CX(16)+CX(17)+CX(18)+P(19)) \quad (4.50)$$
$$*(CX(16)+CX(17)+CX(18)+CX(19)),$$

$$G(6)=P(12)*(CX(12)+P(13))$$
$$*(CX(12)+CX(13)+P(14))$$
$$*(CX(12)+CX(13)+CX(14)+P(15)) \quad (4.51)$$
$$*(CX(12)+CX(13)+CX(14)+CX(15)),$$

$$G(7)=P(8)\ *(CX(8)+P(9))$$
$$*(CX(8)+CX(9)+P(10))$$
$$*(CX(8)+CX(9)+CX(10)+P(11)) \quad (4.52)$$
$$*(CX(8)+CX(9)+CX(10)+CX(11)),$$

$$G(8)=P(4)\ *(CX(4)+P(5))$$
$$*(CX(4)+CX(5)+P(6))$$
$$*(CX(3)+CX(4)+CX(5)+P(7)) \quad (4.53)$$
$$*(CX(4)+CX(5)+CX(6)+CX(7)),$$

$$G(9)=P(Q)\ *(CX(Q)+P(1))$$
$$*(CX(Q)+CX(1)+P(2))$$
$$*(CX(Q)+CX(1)+CX(2)+P(3)) \quad (4.54)$$
$$*(CX(Q)+CX(1)+CX(2)+CX(3)),$$

Ha a PG(i) csoportterjedés 1, az azt jelenti, hogy az i-edik csoport továbbítja az IG(i) i-edik bemeneticsoport-átvitelt, ha ez utóbbi előfordul. Így PG(i)*IG(i) tag azt fejezi ki, hogy az IG(i) bemeneticsoport-átvitel i-edik csoporton keresztül terjedt. A PG(i) értéke akkor 1, ha az első csoport négy szakaszán a P(35), P(34), P(33) és P(32) egyaránt 1. Így a következőket kapjuk:

$$PG(1)=P(32)*P(33)*P(34)*P(35) \quad (4.55)$$

Hasonlóképpen a többi csoportban a csoportterjedések:

$$PG(2)=P(28)*P(29)*P(30)*P(31)$$
$$PG(3)=P(24)*P(25)*P(26)*P(27)$$
$$PG(4)=P(20)*P(21)*P(22)*P(23)$$
$$PG(5)=P(16)*P(17)*P(18)*P(19)$$
$$PG(6)=P(12)*P(13)*P(14)*P(15) \quad (4.56)$$
$$PG(7)=P(8)*P(9)*P(10)*P(11)$$
$$PG(8)=P(4)*P(5)*P(6)*P(7)$$
$$PG(9)=P(Q)*P(1)*P(2)*P(3)$$

A bemeneticsoport-átviteleket a csoportot megvalósító logikai hálózat generálja, amint az a 4.16. ábrán látható. Az 1. csoportban az IG(1) bemeneticsoport-átvitel egyúttal a teljes összeadó 35-ik szakaszának C(35) bemeneti átvitele is. Így:

$$IG(1)=C(35)$$

A 2. csoport IG(2) bemeneticsoport-átvitele lehet az első csoportban generált G(1) átvitel, vagy az 1. csoporton keresztül terjedő IG(1) bemeneticsoport-átvitel. Így a következőket kapjuk:

$$IG(2)=G(1)+IG(1)*PG(1)$$

A többi bemeneticsoport-átvitel hasonlóképpen vezethető le, és a (4.25)...(4.32)-ben látható.

Az egyes G(i) csoportátvitelekhez tartozó szint a (4.45)-ből és a (4-47)...(4-54)-ből értékelhető: ez láthatóan 4. A PG(i) csoportterjedésekhez tartozó szint a (4.55)-ből és a (4.56)-ból értékelhető: ez 2. Ily módon, az IG(1), IG(2), ..., IG(10) bemeneticsoport-átvitelekhez tartozó szintek a (4.23)...(4.32)-ből számíthatók ki: ezek 0, 4, ..., 20. A bemeneticsoport-átvitelek, azokkal a szintekkel együtt, amelyeken keresztül terjedhetnek, a 4.1. táblázat harmadik és negyedik oszlopában láthatók.

4.4.5. Szekcióátvitelek

Az alábbiakban levezetjük az S(i) szekcióátvitelekhez, PS(i) szekcióterjedésekhez és IS(i) bemenetiszekció-átvitelekhez tartozó Boole-függvényeket. Az első szekció az 1., 2. és 3. csoportot tartalmazza. Az S(1) szekcióátvitel a szekción belül generálódik, miközben a bemenetiszekció-átvitelt elhanyagolják. Így S(1) lehet a harmadik csoportból származó G(3) csoportátvitel, vagy a második csoportból jövő G(2) csoportátvitel, amely keresztül terjedt a harmadik csoporton, vagy az első csoportból jövő G(1) csoportátvitel, amely az első és a második csoporton terjedt keresztül. Tehát az első szekcióhoz tartozó Boole-függvény:

$$S(1)=G(3)+G(2)*PG(2)*PG(3)+G(1)*PG(1)*PG(2)*PG(3) \tag{4.57}$$

A többi szekcióátvitelt hasonlóképpen lehet levezetni. Ezek:

$$S(2)=G(6)+G(5)*PG(5)*PG(6)+G(4)*PG(4)*PG(5)*PG(6) \tag{4.58}$$

és

$$S(3)=G(9)+G(8)*PG(8)*PG(9)+G(7)*PG(7)*PG(8)*PG(9) \tag{4.59}$$

Ha a PS(i) szekcióterjedés 1, ez azt mutatja, hogy az i-edik szekción áthalad az IS(i) i-edik bemenetiszekció-átvitel, ha az utóbbi előfordul. Így, $PS(i)*IS(i)$ kifejezés azt mutatja, hogy IS(i) bemenetiszekció-átvitel áthaladt az i-edik szekción. PS(i) 1-gyel egyenlő, ha az első szekció PG(1), PG(2) és PG(3) csoportterjedése egyaránt 1. Tehát, a következőt kapjuk:

$$PS(1)=PG(1)*PG(2)*PG(3) \tag{4.60}$$

Hasonlóképpen, a többi szekcióban a szekció terjedés:

$$PS(2)=PG(4)*PG(5)*PG(6) \tag{4.61}$$
$$PS(3)=PG(7)*PG(8)*PG(9) \tag{4.62}$$

A bemenetiszekció-átviteleket szekciófelépítő logikai hálózat generálja, amint az a 4.17. ábrán látható. Az 1. szekció IS(1) szekcióátvitele megyegyezik a teljesösszeadó harmincötödik szakaszának C(35) bemeneti átvitelével. Tehát:

$$IS(1)=C(35)$$

A 2. szekció IS(2) bemenetiszekció-átvitele tartalmazza az első szekcióban generált S(1) szekció átvitelt vagy azt az IS(1) bemenetiszekció-átvitelt, amely az 1. szekción keresztül terjedt. Így a következőt kapjuk:

$$IS(2)=S(1)+IS(1)*PS(1)$$

A többi, hasonlóképpen levezethető bemenetiszekció-átvitel a (4.35)-ben és (4-36)-ban látható.

Az egyes S(i) szekcióátvitelekhez tartozó szinteket a (4.57), (4.58) és (4.59) alapján lehet értékelni: számuk megállapíthatóan 6. A szekcióterjedésekhez tartozó szinteket a (4.60), (4.61) és (4.62) alapján lehet számítani: ezek száma 3. Az IS(1), ..., IS(4) bemenetiszekció-átvitelekhez tartozó szintek számát a (4.33)...(4.36)-ból lehet értékelni: ezek száma 0, 6, 8 és 10. A bemenetiszekció-átviteleket, amelyek az IG(1), IG(4), IG(7) és IG(10) átviteleket helyettesítik, azokkal a szintekkel együttvéve, amelyeken keresztül ez utóbbiak terjedhetnek, a 4.1. táblázat ötödik és hatodik oszlopában találhatjuk meg. Az összegbitekhez tartozó szinteket a (4.21)-ből számíthatjuk ki: ezeket a 4.1. táblázat hetedik és nyolcadik oszlopában láthatjuk.

4.5.
Tervezési szempontok

Ebben a szakaszban két problémakört tárgyalunk: a kétlépcsős regiszter alkalmazását, valamint egyéb szükséges gépi utasítások bevezetését, összeadási, kivonási, szorzási és osztási utasítások kiegészítésére.

4.5.1.
Kétlépcsős regiszterek

A (4.11), (4.12) és (4.13) utasításokkal leírt művelet sorozatokban elég sok regiszterátvitel szerepel. Ezek az átvitelek olyan regiszterek között mennek végbe, amelyeket felépítő flip-flopok bemenetén vagy kimenetén lehet késleltetés. A gyakorlatban ezek a késleltetések korlátozzák a regiszter átviteli sebességét és megszabják a minimális óraperiódusidőt. A regiszterátviteli sebesség javításának egyik módja a kétlépcsős regiszterek alkalmazása.

A kétlépcsős regiszter két egyforma regiszterből áll, de ezek funkcionálisan egy regisztert alkotnak. Például helyettesítse az AC regisztert egy kétlépcsős regiszter, ezeket nevezzük AC1- és AC2-nek. A szám rendszerint az AC1 regiszterben van tárolva, azonban átmenetileg először az AC2-ben tárolódik. Mármost vizsgáljuk meg a következő végrehajtási utasítást, amelyet a (4.11) leírásból vettünk:

```
(SI * ADKIV * K(7) * P(1))   AC(R,Q,M) ← 0 — AC(Q,M)
                             add2 0 — 0 — TR(M) — 0          (4.63)
```

Kétlépcsős regiszter esetén az összeget először az AC2 regiszter tárolja és ezután kerül az AC1 regiszterbe. Az átvitelek pontos leírásához a következőket kell felírnunk:

```
Register,                    AC1(S,R,Q,1 — 35),
                             AC2(S,Q,R,1 — 35)               (4.64)
Clock,                       P(1 — 3)
(EI * ADKIV * K(7) * P(1))   AC2(R,Q,M) ← 0 — AC1(Q,M)
                             add2 0 — 0 — TR(M) — 0,
(EI * ADKIV * K(7) * P(2))   AC1(R,Q,M) ← AC2(R,Q,M),
(P(3))                       D ← countup D
```

A fenti leírásban háromfázisú órajelgenerátort alkalmaztunk. Az első fázis alatt történik az összeadás. A második fázisban az AC2-ben levő szót átvisszük az AC1 regiszterbe. Ezt a két átvitelt olyan gyorsan lehet végezni, amennyire azt az elemek lehetővé teszik anélkül, hogy késleltetésekre lenne szükség a regiszterek bemenetein vagy kimenetein. A harmadik fázisban a D ütemregiszter inkrementálódik.

Háromfázisú órajelgenerátor alkalmazása helyett megtarthatjuk a két fázisú órajelgenerátort, de késleltetésekkel egybekapcsolva. Legyenek a késleltetések

```
Delay,                       KSL                             (4.65)
```

A késleltetés nagysága egyenlő vagy kisebb lehet a két órajelfázis — P(1) és P(2) — közötti időnél. A késleltetőelem figyelembevételével a (4.64) leírást a következőképpen helyesbíthetjük:

```
Register,                          AC1(S,R,Q,1—35),
                                   AC2(S,R,Q,1—35),
Clock,                             P(1—2)                          (4.66)
/EI * ADKIV * K(7) * P(1)/         AC2(R,Q,M) ← 0—AC1(Q,M)
                                   add2 0—0—TR(M)—0,
/KSL(EI * ADKIV * K(7) * P(1)/     AC1(R,Q,M) ← AC2(R,Q,M)
/P(2)/                             D ← countup D,
```

A fenti, az add2 mikroművelethez tartozó vezérlőjel először késleltetve van és ezek után kerül felhasználásra vezérlőjelként az AC2 regiszterből az AC1 regiszterbe való átvitelhez. Ezáltal az átviteli mikroművelet nem akadályozza a számlálót növelő mikroműveletet a P(2) órajelfázisban.

4.5.2. Fixpontos aritmetikai utasítás

Egycímes utasításrendszerű gépnél egy műveleti kóddal és operandus címmel ellátott utasítás szükséges az aritmetikai egység vezérléséhez, hogy végrehajtsa az összeadási, kivonási, szorzási vagy osztási műveletsorozatot. Tételezzük föl, hogy az utasítás csak műveleti kódot és egy operandus címet tartalmaz. Ekkor az utasítások alakja ADD y, SUB y, MPY y és DIV y, ahol y határozza meg azt a tárcímet, ahol rendre ADD esetén az összeadandó, SUB esetén a kivonandó, MPY esetén a szorzandó, és DIV esetén az osztandó helyezkedik el. Ezek az utasítások azonban magukban nem elegendők, további utasításokra van szükség.

A tárban levő két szám összeadásához két utasításra van szükség; ezek:

— a CLA (**CL**ear and **A**dd: törlés és összeadás) és
— az ADD.

A CLA utasítás viszi át az egyik összeadandót a tárból az akkumulátorba és az ADD utasítás adja hozzá a másik összeadandót (az utóbbit a tárból véve) az akkumlátor tartalmához. Ha az eredményt a tárban akarjuk elhelyezni, a STO (store—tárolás) utasításra is szükség van. Egyik számnak a másikból való kivonásához (ha mindkét szám a tárban van) és az eredménynek a tárban való elhelyezéséhez három utasítás szükséges ezek:

— CLA y,
— SUB y+1,
— STO y+2.

Három olyan további utasítás, amelyet kényelmesen lehet megvalósítani és gyakran hasznos lehet:

— a CLS (**CL**ear and **S**ubtract: törlés és kivonás,
— az ADM (**AD**d **M**agnitude: érték összeadás) és
— az SBM (**S**u**B**tract **M**agnitude: érték kivonás).

A CLS megegyezik a CLA utasítással, kivéve, hogy az összeadó előjelét először ellentétesre változtatják. Az ADM és SBM utasítások ugyanazok, mint az ADD és a SUB, kivéve, hogy a szám először a tárban pozitív előjelet kap.

Két olyan szám szorzásához, amelyek mindegyike a tárban helyezkedik el, két utasítás, az LDQ (**L**oa**D** M**Q**: MQ betöltése) és az MPY szükséges. Az LDQ utasítás tölti be a szorzót az MQ regiszterbe. Az MPY utasítás törli az akkumulátort, a szorzandót át-

viszi a TR regiszterbe és kezdeményezi a szorzási műveletsorozatot. A szorzat az AC és TR regiszterekbe kerül oly módon, hogy a nagyobb helyiértékű része az akkumulátorban és a kisebb helyiértékű része az MQ regiszterben helyezkedik el. Ha mindkét részt a tárban kell tárolni, két utasításra van szükség: STO y, és STQ y+1 (**STo**re **MQ**: MQ tárolása). A STO utasítás beviszi az MQ regiszter tartalmát a tárba, az y+1 címre helyezve el azt.

A léptetőutasítások a helyiérték-változtatásokkor hasznosak: ezek mozgatják az AC regiszterben vagy az AC(M)—MQ(M) regisztersorban levő szám bináris pontját. Az ALS (**A**ccumulator **L**eft **S**hift: akkumulátor balra lép) vagy ARS (**A**ccumulator **R**ight **L**eft: akkumulátor jobbra lép) utasítások balra, ill. jobbra léptetik az AC(M)részregiszter tartalmát; a lépésszámot a cím határozza meg. Ezeknél az utasításoknál a cím lépésszámot reprezentál. Az LLS (**L**ong **L**eft **S**hift: hosszú balra léptetés) és LRS (**L**ong **R**ight **S**hift: hosszú jobbra léptetés) utasítások balra, ill. jobbra léptetik az AC(M)—MQ(M) regisztersor tartalmát; a lépésszámot (bitpozíciók számát) a cím határozza meg. Ha nincs szükség kétszeres hosszúságú szorzatra, a RND ((**RouND**: kerekítés) utasítást lehet használni, amely a szorzat nagyobb helyiértékű részét kerekíti. A kerekítés azt az egyszerű szabályt használhatja, hogy az AC(M) tartalmát 1 bittel növeli, ha az MQ(1) bit értéke 1. A RND utasítás helyett használható az MPR (**M**ulti**P**ly and **R**ound: szorzás és kerekítés) utasítás. Ez az utasítás megegyezik a MYP utasítással, kivéve, hogy az AC(M)-ben levő nagyobb helyiérékű részt a szorzás után kerekíti.

A DIV utasítás az AC(M)—MQ(M) regiszterben levő osztandóra vonatkozik, amelynek hossza esetünkben 70 bit. Ehhez az osztás előtt fel kell tölteni az AC és az MQ regisztert a CLA és a LDQ utasításokkal, továbbá ha szükséges, a léptetőutasításokkal el kell végezni a helyiérték-változtatásokat. Osztás után az MQ regiszterben levő hányadot az STQ, és az AC regiszterben levő maradékot a STO utasítással lehet elhelyezni a tárban. Ha osztáskor túlcsordulás következik be, a DVOV indikátor 1 értéket kap és a számítógép működése leáll. Azonban tulajdonképpen felesleges leállítani a számítógép működését. Ezért az osztást két utasítással, a DVH (**DiV**ide or **H**alt: oszt vagy leáll) és a DVP (**DiV**ide or **P**roceed: oszt vagy folytatja) lehet elvégezni. Az első leállítja a számítógép működését ha osztási túlcsordulás következik be, míg a második nem.

4.6.
Az aritmetikai egység mikroprogramozása

A párhuzamos, bináris, fixpontos aritmetikai egységet a (4.1)...(4.4) utasításokban szereplő struktúrával és a 4.6,...,4.9., 4.11 és 4.12. ábrákon levő folyamatábrákkal írtuk le. Az így leírt aritmetikai egységet huzalozott logikai vezérlés valósította meg. A huzalozott logikai vezérlés struktúráját a (4.10) utasításban, az aritmetikai műveletek sorozatát pedig a (4.11)...(4.13) utasításokban írtuk le.

Ebben a szakaszban a mikroprogramozott vezérlés alkalmazását írjuk le az aritmetikai egység működtetésére. Bemutatjuk a vezérlés struktúráját, a vezérlőjeleket, a mikroutasítás formátumát, a sorrendi leírásokat és a mikroprogramot.

4.6.1.
Mikroprogramozott vezérlés struktúrája

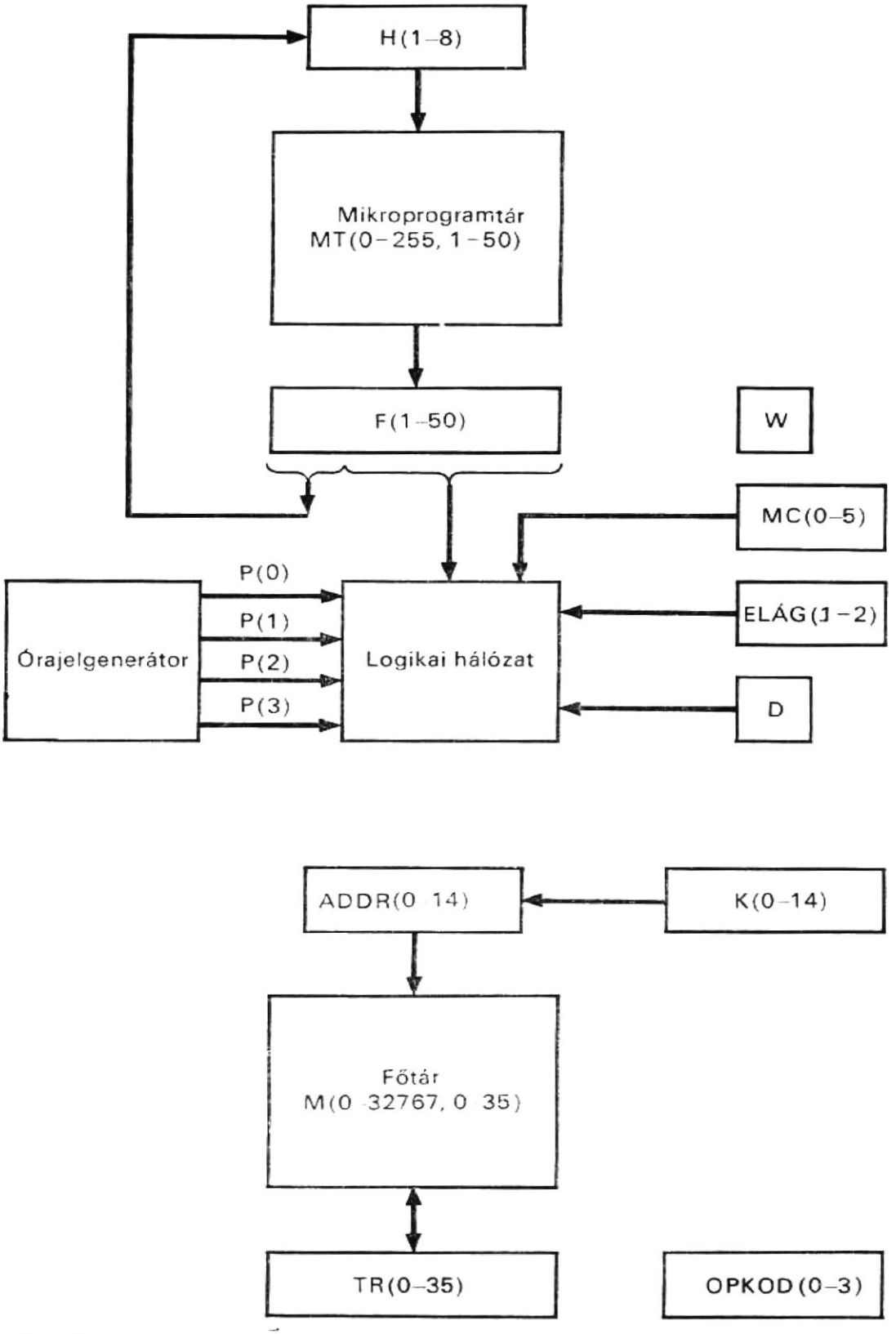
A mikroprogramozott vezérlés struktúrája a 4.18. ábrán levő logikai hatásvázlaton látható. Az MT mikroprogramtár kapacitása 256×36-bites szó és a H címregiszter, valamint az F tartalomregiszter tartozik hozzá. Az ELÁG regiszter két bitje végzi az elágazás vezérlését. A négyfázisú P(0—3) órajelgenerátor, az egybites D regiszter és a 6-bites MC regiszter generálják a vezérlőjeleket. Az OPKOD regiszter tárolja az utasítások műveletikód-részét. Az M főtárnak 32 768×36-bites szó a kapacitása és az ADDR címregiszter és a TR tartalomregiszter tartozik hozzá. A K regiszter az operandus címet tárolja. Ezt a struktúrát az alábbi utasításokkal írhatjuk le:

Comment, mikroprogramozott vezérlés struktúrája[8] (4.67)

```
Register,    H(0—7),                    $mikroprogramtár címregisztere
             F(0—35),                   $mikroutasítás-regiszter
             MC(0—5),                   $főtárciklus-regiszter
             D,                         $mikroprogramtár-ciklus várakozás-
                                          regiszter
             OPKOD(0—3),                $műveletikód-regiszter
             K(0—14),                   $operandus cím-regiszter
             ADDR(0—14),                $főtár címregisztere
             TR(0—35),                  $főtár tartalomregisztere
Subregister, F(ADKIV)=F(0—7),           $az F regiszter címmezője
Memory,      MT(H)=MT(0—255, 0—35),     $mikroprogramtár
             M(ADDR)=M(0—32767,0—       35), $főtár
Clock,       P(0—3),                    $négyfázisú órajelgenerátor
Block,       DSET (IF (MC(0)=MC(1)+MC(2)+MC(3)+MC(4)=1)
                  THEN (D←1,) IF (MC(5)=1)
                  THEN (D←0),
             MQSET (IF (MQ(34,35)=00)
                  THEN (MQ(34,35)←01, C←0),
                  IF (MQ(34,35)=01)
                  THEN (MQ(34,35)←10, C←0),
                  IF (MQ(34,35)=10)
                  THEN (MQ(34,35)←11),
                  IF (MQ(34,35)=11)
                  THEN (MQ(34,35)←00)),
             MQELÁG (IF (MQ(34,35)=01) THEN (AC(R,Q,M)←
                    AC(R,Q,M) add2 0—0—TR(M)—0),
                  IF (MQ(34,35)=10) THEN (AC(R,Q,M)←
                    AC(R,Q,M) add2 0—0—TR(M)—0—0),
                  IF (MQ(34,35)=11) THEN (AC(R,Q,M)←
                    0—0—AC(M) add2—1—1—TR(M)'—1, C←1)),
```

A fenti struktúrában a blokkutasítás három mikroutasítás-csoport nevét deklarálja, ezek: DSET, MQSET és MQELÁG. A DSET blokk a D regisztert 1-be vagy 0-ba vivő feltételes mikroműveletet reprezentálja. Az MQSET blokk azon mikroutasítás-csoportot képviseli, amely a szorzási műveletsorozatban szükséges korrekciót hajtja végre. Az MQELÁG blokk azon mikroutasítások csoportjából áll, amelyek az MQ(34,35)

[8] A struktúraleírásában, a (4.67)-ben hiányzik az ELÁG regiszter deklarációja. Tehát pl. a TR regiszter után következnék: ELÁG (1—2), $elágaztatásregiszter *(Fordító)*



4.18. ábra
Mikroprogramozott vezérlésű struktúra a párhuzamos, bináris, fixpontos aritmetikai egység realizálásához

biteket vizsgálják, és ezt követően vagy nem végzik el az összeadást, vagy összeadást vagy kivonást végeznek úgy, ahogy azt a korábbiakban a szorzási sorozatban leírtuk. Az MQSET és az MQELÁG blokkok által képviselt mikroműveletek a 4.9. folyamatábrán bemutatottaknak felelnek meg.

4.6.2.
Ütemezés és vezérlőjelek

Minden főtárciklust úgy választottunk meg, hogy hat mikroprogramtár-ciklussal essék egybe, míg egy mikroprogramtár-ciklus azonos egy órajelciklussal. Ezért, míg minden mikroprogramtár-ciklusban négy lépés van, az egyes főtárciklusok 24 lépésből állnak. A vezérlőjeleket ehhez a 24 lépéshez a következő cimkesorozat írja le (kiegészítésképpen az indítási cimkékhez):

Comment, vezérlőjelek egy főtárciklus alatt (4.68)

```
/START(ON)*P(3)/   MC←8,
/MC(0)*P(0)/                        $mindkét tárciklus kezdete
/MC(0)*P(1)/
/MC(0)*P(2)/
/P(3)/             MC←cir MC        $a mikroprogramtár-ciklus vége
/MC(1)*P(0)/                        $a mikroprogramtár-ciklus kezdete
/MC(1)*P(1)/
/MC(1)*P(2)/
/P(3)/             MC←cir MC        $a mikroprogramtár-ciklus vége
.................
.................
/MC(5)*P(0)/                        $a mikroprogramtár-ciklus kezdete
/MC(5)*P(1)/
/MC(5)*P(2)/
/P(3)/             MC←cir MC        $mindkét tárciklus vége
```

A fenti cimkékben minden mikroprogramtár-ciklusban a négy lépést a P(0—3) órajel négy fázisa ütemezi, míg az egyes főtárciklusokban a hat mikroprogramtár-ciklust az MC(0—5) gyűrűs számláló hat állapota biztosítja, ezért az MC-t ciklikusan jobbra léptetik az egyes mikroprogramtár-ciklusok utolsó, P(3) órajelfázisában.

Az egyes főtárciklusok alatt egy szót olvasnak ki vagy írnak be az M főtárból (ba). Tételezzük föl, hogy a K regiszterben levő operanduscím átvitele az ADDR főtár címregiszterébe, valamint a főtár olvasási vagy írási műveletének kezdeményezése a második lépés alatt megy végbe. Az olvasási műveletnél a szó a főtár TR tartalomregiszterében a 14-ik lépés folyamán áll rendelkezésre. Tehát a következőket kapjuk:

```
/MC(0)*P(1)/       ADDR←K,          (4.69)
/MC(2)*P(1)/       TR←M(ADDR),
```

A négy lépés vezérlőjeleit az egyes mikroprogramtár-ciklusokban a következő cimkék írják le:

Comment, vezérlőjelek egy mikroprogramtár-ciklus alatt (4.70)

```
/P(0)*D'/    $a mikroprogramtár-ciklus kezdete
/P(1)*D'/
/P(2)*D'/
/P(3)*D'/    $a mikroprogramtár-ciklus vége
```

Ha a D regiszter tartalma 0, a fenti cimkék „megjelennek", tehát hatásuk érvényesül, egyébként pedig „eltűnnek". Tehát a D regiszter vezérli minden mikroprogramtár-ciklusban a lépések haladását vagy leállását.

Az egyes mikroprogramtár-ciklusokban egy mikroutasítás (szó) kerül kiolvasásra a mikroprogramtárból. Tételezzük föl, hogy a mikroprogramtár címének a H regisz-

4.2. táblázat

Mikroutasítás formátuma

Vezérlőbit	Órajelfázis	Mikroműveletek
F(0—7)		mikroprogramtárcím
F(8)	P(0)	F←C(H)
F(9)	P(1)	IF (OPKÓD=1) THEN (TR(S)←TR(S)′), AC(R, Q)←0,
F(9)	P(2)	IF (TR(S)≠AC(S)) THEN (AC(M)←AC(M)′, ELÁG(1)←1) ELSE (ELÁG(1)←0),
F(9)	P(3)	AC(R,Q,M)←0—AC(Q, M) add2 0—0—TR(M)—0,
F(10)	P(3)	IF (ELÁG(1)=1) THEN (H←F(ADKIV)) ELSE (H←countup H),
F(11)	P(3)	ELÁG←0,
F(12)	P(1)	IF (AC(Q)=1) THEN (ADOV←1), IF (AC(Q, M)=0) THEN (AC(S)←0),
F(13)	P(3)	H←F(ADKIV),
F(14)	P(3)	DO DSET,
F(15)	P(1)	IF (AC(Q)=1) THEN (AC(R,Q,M(←1 add2 0—AC(Q, M)—0, ELÁG(1)←1) ELSE (ELÁG(1)←0),
F(15)	P(2)	IF (ELÁG(1)=1) THEN (AC(Q)←0, AC(S)←AC(S)′) ELSE (IF (AC(M)=$37...7_8$) THEN (AC←0) ELSE (AC(M)←AC(M)′)),
F(16)	P(1)	IF (TR(S)=MQ(S)) THEN (AC(S)...0, MQ(S)←0) ELSE (AC(S)←1, MQ(S)←1), AC(R,Q,M)←0,
F(17)	P(2)	IF((TR(M)≠0)*(MQ(M)≠0)) THEN (LSZ←18,C←0,MQ(S)←0, ELÁG(1)←0), ELSE (MQ←0, AC(S)←1, ELÁG(1)←1),
F(18)	P(3)	IF (ELÁG(1)=1) THEN (DO DSET),
F(19)	P(1)	IF (C=1) THEN (DO MQSET),
F(19)	P(2)	DO MQELÁG,
F(20)	P(3)	AC(2—35)—MQ←AC(Q, M)—MQ(S, 1—33), AC(Q, 1)←AC(R)—AC(R),
F(20)	P(1)	LSZ←countdn LSZ,
F(21)	P(3)	IF (LSZ=0) THEN (H←countup H),
F(22)	P(1)	AC(M)←AC(2—35)—MQ(S),
F(22)	P(2)	MQ(S)←AC(S),

4.2. táblázat folytatása

F(23)	P(1)	AC(R,Q,M)←0—0—AC(M)′, IF (Z(Q)≠1) THEN (DVOV←1, ELÁG(1)←1) ELSE (ELÁG(2)←1),
F(23)	P(2)	IF (ELÁG(2)=1) THEN (MQ(S)←TR(S) AC(S) ⊙ LSZ←35),
F(24)	P(1)	AC(M)←MQ(M)←AC(2—35)—MQ(1)′—MQ(2—35)—0,
F(24)	P(2)	IF (Z(Q)≠1) THEN (MQ(35)←1, AC(R, Q, M)←0—0—AC(M) add2 0—0—TR(M)—0)
F(25)	P(1)	AC(M)←AC(M)′
F(26)	MC(0)∗P(1)	ADDR←K,
F(26)	MC(2)∗P(1)	TR←M(ADDR),
F(27)	MC(2)∗P(3)	H←countup H,
F(28—35)		nincs felhasználva

terbe való átvitele, valamint a mikroprogramtárból való kiolvasás kezdeményezése egyaránt a megelőző mikroprogramtár-ciklus P(3) órajelfázisában történik, és a mikroutasítás (szó) az F tartalomregiszterben az aktuális tárciklus P(0) órajelfázisában áll rendelkezésre, vagyis:

```
/P(3)*D'/    H←countup H,      $a megelőző mikroprogramtár-ciklus vége
/P(0)*D'/    F←MC(H),          $az aktuális mikroprogramtár-ciklus kezdete
/P(1)*D'/                                                        (4.71)
/P(2)*D'/
```

Az F regiszterben levő vezérlőbitek által aktivált mikroműveletek az aktuális mikroprogramtár-ciklus P(1—3) órajelfázisaiban kerülnek végrehajtásra.

4.6.3. Mikroutasítás-formátum

A 4.2. táblázatban láthatjuk a mikroutasítás formátumát. Az F regiszterben levő mikroutasítás 36 bitje három csoportra oszlik: az F(0—7) mezőre, amely a mikroprogramtár-címet, az F(8—27) mezőre, amely a vezérlőbiteket tartalmazza, valamint az F(28—35) mezőre, amely nincs felhasználva. Az F(0—7) mező elágaztatási címmel látja el az egyes mikroutasításokat. Az F(8—27) mező mindegyik bitje arra van kijelölve, hogy egy vagy több mikroműveletet vezéreljen, amelyek egyetlen lépésként vagy sorozatként fordulhatnak elő. Ezek azok a mikroműveletek, amelyek az aritmetikai műveletsorozatok folyamatábráin találhatók meg. A 4.2. táblázat feltünteti az egyes vezérlőbitekhez rendelt órajelfázist is. Ezt a hozzárendelést mutatjuk be a következő pontban.

4.6.4. Az aritmetikai sorozatok mikroprogramozása

A mikroutasítás-formátummal (l. 4.2. táblázat) a (4.68) és (4.70)-ben leírt vezérlőjeleket hozzárendeljük az aritmetikai sorozatok folyamatábráin levő mikroműveletekhez. Ennek a hozzárendelésnek három része van. Az első rész a vezérlőjelek hozzárendelése a kezdőérték-adáshoz, valamint az időzítő- és vezérlőjelek generálásához; ez a rész nem tartozik a mikroprogramhoz. A második rész a folyamatábrákon levő mikroműveleteket egy vagy több mikroutasításhoz (vagy vezérlőszóhoz) rendeli hozzá, míg a harmadik rész vezérlőjeleket jelöl ki mindegyik mikroművelethez az egyes mikroutasításokban.

Az aritmetikai egység leírásának első része vezérlőjeleket jelöl ki három mikroművelet-csoporthoz. Az első csoport az ütemezőjeleket generálja az egyes főtárciklusokban levő hat mikroprogramtár-ciklushoz. A második csoport kezdeményezi a kívánt mikroutasítás lehívását a mikroprogramtárból. A harmadik csoport helyezi el a kívánt címet a mikroprogramtár címregiszterébe. Feltételezzük, hogy az első mikroprogramtárcím az összeadási—kivonási sorozathoz, a szorzási sorozathoz és az osztási sorozathoz sorrendben a következő lesz: 0, 6, ill. 12. Ezek a mikroműveletek és a hozzájuk rendelt vezérlőjelek az alábbiakban láthatók:

```
Comment, a párhuzamos, bináris aritmetikai egység mikroprogramozása   (4.72)
/START(ON)*P(2)/  MC←40₈, F←0,
/P(3)/            MC←cir MC,
```

```
/P(3)/          D←0, F(8)←1,
/P(3)/          IF (OPKOD=0+1) THEN (H←0),
                IF (OPKOD=2) THEN (H←6),
                IF (OPKOD=3) THEN (H←12),
```

A fenti utasításokban az összeadás, kivonás, szorzás és osztás műveleti kódjai sorrendben a következők: 0, 1, 2, ill. 3. A vezérlőjelek kijelölését a második és a harmadik részhez az alábbiakban mutatjuk be, mindhárom sorozathoz.

Az összeadás—kivonás sorozathoz négy mikroutasítás tartozik:

— az AS/1,
— az AS/2,
— az AS/3 és
— az AS/4.

Az AS/1 mikroutasítás viszi át a K regiszterben levő operandus címét az ADDR regiszterbe és olvassa ki az operandust az M főtárból. A 4.6. és 4.7. folyamatábrákon levő mikroműveletek három mikroutasításhoz vannak hozzárendelve: a kezdetiérték-adás az AS2 mikroutasításhoz, az összeadási rész az AS3 mikroutasításhoz és a kivonási rész az AS4 mikroutasításhoz. A vezérlőjelek hozzárendelését ezen folyamatábrák mikroműveleteihez az alábbiakban mutatjuk be:

```
Comment, az összeadás—kivonás sorozat mikroprogramozása                (4.73)
Comment, AS1 mikroutasítás

/D'*P(0)*F(8)/      F←MT(H),
/MC(0)*P(1)*F(26)/  ADDR←K,
/MC(2)*P(1)*F(26)/  TR←M(ADDR),
/MC(2)*P(3)*F(27)/  H←countup H.

Comment, AS2 mikroutasítás                                             (4.74)

/D'*P(0)*F(8)/      F←MT(H),
/D'*P(1)*F(9)/      IF (OPKOD=1) THEN (TR(S)←TR(S)'), AC(R,Q)←0,
/D'*P(2)*F(9)/      IF (TR(S)≠AC(S)) THEN (AC(M)←AC(M)', ELÁG(1)←1)
                                     ELSE (ELÁG(1)←0),
/D'*P(3)*F(9)/      AC(R,Q,M)←0—AC(Q,M) add2 0—0—TR(M)—0,
/D'*P(3)*F(10)/     IF (ELÁG(1)=1) THEN (H←F(ADKIV))
                                   ELSE (H←countup H), $F(ADKIV)=AS4
/D'*P(3)*F(11)/     ELÁG←0,

Comment, AS3 mikroutasítás                                             (4.75)

/D'*P(0)*F(8)/      F←MT(H),
/D'*P(1)*F(12)/     IF (AC(Q)=1) THEN (ADOV←1),
/D'*P(1)*F(12)/     IF (AC(Q,M)=0) THEN (AC(S)←0),
/D'*P(3)*F(13)/     H←F(ADKIV),                    $F(ADKIV)=exit cím
/D'*P(3)*F(14)/     DO DSET,

Comment, AS4 mikroutasítás                                             (4.76)

/D'*P(0)*F(8)/      F←MT(H),
/D'*P(1)*F(15)/     IF (AC(Q)=1) THEN (AC(R,Q,M)←1 add2 0—AC(Q,M)—0,
                                   ELÁG(1)←1)
                                   ELSE (ELÁG(1)←0),
/D'*P(2)*F(15)/      IF (ELÁG(1)=1)
                    THEN (AC(Q)←0, AC(S)←AC(S)')
                    ELSE (IF (AC(M)=37...7₈)
```

```
                        THEN (AC←0)
                        ELSE (AC(M)←AC(M)')),
/D'*P(3)*F(13)/         H←F(ADKIV)              $F(ADKIV)=exit cím
/D'*P(3)*F(11)/         ELÁG←0
/D'*P(3)*F(14)/         DO DSET
```

A fentiek szerint az AS2, AS3 és AS4 mikroutasításokban levő első mikroművelet kiolvassa a mikroprogramtárból a mikroutasításokat. Az ELÁG(1) bitet arra használjuk az AS2 mikroutasításban, hogy elágaztatást hajtson végre az összeadás—kivonás sorozat összeadási vagy kivonási részéhez. Az AS4 mikroutasításban az ELÁG(1) bit szerepe az, hogy elágaztatás jöjjön létre attól függően, hogy a különbség negatív- vagy pozitív-e. Az AS2 és AS4 mikroutasításban a D regiszter 1 értéket kap, ha az aktuális mikroprogramtár-ciklus nem az MC(5) tárciklus, és 0 lesz az értéke, ha igen, vagyis ha MC(5)=1. A D regisztert 1-be vagy 0-ba vivő feltételes értékadására vonatkozó mikroműveleteket a DO DSET mikroutasítás írja le. Ily módon az összeadás—kivonás sorozat a főtárciklus végén fejeződik be.

Az összeadás—kivonás sorozat végrehajtása egy főtárciklust (azaz hat, MC(0)...MC(5)-tel vezérelt mikroprogramtár-ciklust) vesz igénybe. Az AS1 mikroutasításban levő mikroműveleteket az MC(0) és MC(2) mikroprogramtár-ciklus alatt hajtják végre; az MC(1) mikroprogramtár-ciklus folyamán nem történik mikroművelet. Az AS2, AS3 és AS4 mikroutasításokat az MC(3), MC(4), ill. MC(5) mikroprogramtár-ciklusok alatt hajtják végre.

A szorzási sorozathoz négy mikroutasítás tartozik: M1, M2, M3 és M4. Az M1 mikroutasítás megegyezik a (4.73)-ban leírt utasítással. A 4.8. és 4.9. folyamatábrákon levő mikroműveleteket három mikroutasításhoz rendeljük hozzá: az értékadó részt az M2 mikroutasításhoz, a szorzási részt az M3 mikroutasításhoz és a befejező részt az M4 mikroutasításhoz. A vezérlőjeleknek ezen folyamatábrákon levő mikroműveletekhez való hozzárendelését az alábbiakban mutatjuk be.

```
Comment, a szorzási sorozat mikroprogramozása
```

```
Comment, M1 mikroutasítás                                          (4.77)
/D'*P(0)*F(8)/          F←MT(H),
/MC(0)*P(1)*F(26)/      ADDR←K,
/MC(2)*P(1)*F(26)/      TR←M(ADDR),
/MC(2)*P(3)*F(26)/      H←countup H,
```

```
Comment, M2 mikroművelet                                           (4.78)
/D'*P(0)*F(8)/          F←MT(H),
/D'*P(1)*F(16)/         IF (TR(S)=MQ(S)) THEN (AC(S)←0, MQ(S)←0)
                                         ELSE (AC(S)←1, MQ(S)←1),
                                           AC(R,Q,M)←0,
/D'*P(2)*F(17)/         IF ((TR(M)≠0)*MQ(M)≠0)) THEN (LSZ←18,C←0,
                                                MQ(S)←0, ELÁG(1)←0)
                                              ELSE (MQ←0, AC(S)
                                                ←0, ELÁG(1)←1),
/D'*P(3)*F(10)/         IF (ELÁG(1)=1) THEN (H←F(ADKIV))
                                       ELSE (H←countup H),
                                       $F(ADKIV)=exit cím
/D'*P(3)*F(18)/         IF (ELÁG(1)=1) THEN (DO DSET),
/D'*P(3)*F(11)/         ELÁG←0,
```

```
Comment, M3 mikroutasítás                                          (4.79)
/D'*P(0)*F(8)/          F←MT(H),
```

```
/D'*P(1)*F(19)/     IF (C=1) THEN (DO MQSET),
/D'*P(1)*F(20)/     LSZ←countdn LSZ,
/D'*P(2)*F(19)/     DO MQELÁG,
/D'*P(3)*F(19)/     AC(2—35)—MQ←AC(Q,M)—MQ(S,1—33),
                      AC(Q,1)←AC(R)—AC(R),
/D'*P(3)*F(21)/     IF (LSZ=0) THEN (H←countup H),
```

Comment, M4 mikroutasítás (4.80)

```
/D'*P(0)*F(8)/      F←MT(H),
/D'*P(1)*F(22)/     AC(M)←AC(2—35)—MQ(S),
/D'*P(2)*F(22)/     MQ(S)←AC(S),
/D'*P(3)*F(13)/     H←F(ADKIV),                $F(ADKIV)=exit cím
/D'*P(3)*F(14)/     DO DSET,
```

A fenti leírásban az ELÁG(i) bitet az M2 mikroutasítás elágaztatására használjuk. A szorzási hurok 18-szor végez iterációt, ismételten végrehajtva az M3 mikroutasítást, egészen addig, amíg az LSZ lépésszámláló el nem éri a 0 értéket. Ekkor az M4 mikroutasítás kerül kiolvasásra a mikroprogramtárból és végrehajtódik. A szorzás az M2 vagy az M4 mikroutasítás alatt fejeződhet be. A befejezés ismét a DO DSET mikroutasítás végrehajtásával zárul le.

A szorzási sorozat négy főtárciklust vesz igénybe. Az első ciklus alatt az M1 és M2 mikroutasítás egyszer, az M3 mikroutasítás kétszer hajtódik végre. Az M3 hatszor hajtódik végre a második ciklus, és hatszor a harmadik ciklus folyamán. A negyedik ciklusban négyszer kerül végrehajtásra, hogy teljesítse a szorzási hurok előírt 18 iterációját. Ezután az M4 mikroutasítás végrehajtása következik.

Az osztási sorozathoz ugyancsak négy mikroutasítás tartozik: a D1, D2, D3 és D4. A D1 mikroutasítás megegyezik a (4.73) utasításokban leírtakkal. A 4.11. és 4.12. folyamatábrákon levő mikroműveleteket három mikroutasításhoz rendeljük hozzá: a kezdetiérték-adó részt a D2 mikroutasításhoz, az osztási részt a D3 mikroutasításhoz és a befejező részt a D4 mikroutasításhoz. A vezérlőjelek hozzárendelése ezen folyamatábrákon levő mikroműveletekhez az alábbiakban látható.

Comment, az osztási sorozat mikroprogramozása
Comment, D1 mikroutasítás (4.81)

```
/D'*P(0)*F(8)/      F←MT(H),
/MC(0)*P(1)*F(26)/  ADDR←K,
/MC(2)*P(1)*F(26)/  TR←M(ADDR),
/MC(2)*P(3)*F(27)/  H←countup H,
```

Comment, D2 mikroutasítás (4.82)

```
/D'*P(0)*F(8)/      F←MT(H),
/D'*P(1)*F(23)/     AC(R,Q,M)←0—0—AC(M)',
                    IF (Z(Q)≠1) THEN (DVOV←1, ELÁG(1)←1)
                                ELSE (ELÁG(2)←1),
/D'*P(2)*F(23)/     IF (ELÁG(2)=1) THEN (MQ(S)←TR(S)⊕AC(S), LSZ←35),
/D'*P(3)*F(10)/     IF (ELÁG(1)=1) THEN (H←F(ADKIV))
                                   ELSE (H←countup H),
                                   $F(ADKIV)=exit cím
/D'*P(3)*F(18)/     IF (ELÁG(1)=1) THEN (DO DSET),
/D'*P(3)*F(11)/     ELÁG←0,
```

Comment, D3 mikroutasítás (4.83)

```
/D'*P(0)*F(8)/      F←MT(H),
```

```
/D'*P(1)*F(20)/     LSZ←countdn LSZ,
/D'*P(1)*F(24)/     AC(M)—MQ(M)←AC(2—35)—MQ(1)'—MQ(2—35)—0,
/D'*P(2)*F(24)/     IF (Z(Q)≠1) THEN (MQ(35)←1, AC(R,Q,M)←0—0—AC(M)
                      add2 0—0—TR(M)—0),
/D'*P(3)*F(21)/     IF (LSZ=0) THEN (H←countup H),

Comment, D4 mikroutasítás                                          (4.84)
/D'*P(0)*F(8)/      F←MT(H),
/D'*P(1)*F(25)/     AC(M)←AC(M)',
/D'*P(3)*F(13)/     H←F(ADKIV),          $F(ADKIV)=exit cím
/D'*P(3)*F(14)/     DO DSET,
```

A fenti leírásban az ELÁG (1,2) biteket a D2 mikroutasításban használjuk elágaztatásra. Az osztáshoz szükséges kivonást a D3 mikroutasítás végzi. A kivonás 35-ször kerül végrehajtásra a D3 mikroutasítás ismételt végrehajtásával, amíg csak az LSZ lépésszámláló el nem éri a 0 értéket. Ekkor a D4 mikroutasítás kerül kiolvasásra a mikroprogramtárból és végrehajtódik. Az osztás a D2 vagy a D4 mikroutasítás végrehajtása során fejeződhet be. A befejezés ismét a DO DSET mikroutasítás végrehajtásával zárul le.

Az osztási sorozat hét főtárciklust vesz igénybe. Az első ciklusban a D1 és D2 mikroutasítások egyszer és a D3 mikroutasítás kétszer hajtódik végre. A D3 hatszor kerül végrehajtásra a másodiktól a hatodik ciklusig. A hetedik ciklusban a D3 háromszor hajtódik végre, hogy elvégezze az előírt 35 kivonást és ezután kerül végrehajtásra a D4 mikroutasítás.

4.6.5. Mikroprogram

Az aritmetikai egységhez tartozó mikroprogram a 4.3. táblázatban látható, ahol 12 mikroutasítás található. Az első négy mikroutasítás a 0...3 címeken helyezkedik el, az összeadás—kivonás utasítás sorozathoz. A 6...9 címeken a szorzási sorozat négy mikroutasítása van, a 12...15 címeken pedig az osztási sorozaté. Az exit címének a 16-ot választottuk.

Amint a 3. fejezetben leírtuk, az egyes mikroutasításokban levő 1-eseket azon végrehajtási utasítások cimkéiben szereplő vezérlőbitek határozzák meg, amelyek a mikroutasítást leírják. Ily módon, a 4.3. táblázatban található mikroprogramot a (4.73)... ...(4.84) utasításokból kaptuk.

Feladatok

4.1. Rajzoljuk fel a (4.3) és (4.4) terminálutasításokkal leírt párhuzamos összeadó logikai kapcsolási rajzát!

4.2. Egészítsük ki a 4.5. ábra logikai hatásvázlatát úgy, hogy az a párhuzamos összeadó összeg-kapocspárjait is ábrázolja!

4.3. Írjunk le terminálutasítással egy párhuzamos kivonót és rajzoljuk fel a párhuzamos kivonó logikai kapcsolási rajzát!

4.4. Írjunk le terminálutasítással egy párhuzamos összeadó—kivonót, amely akkor ad össze, ill. von ki, ha a C bemeneti kapocs 0, ill. 1.

4.3. táblázat

Bináris aritmetikai egység mikroprogramja

Mikro-utasítás	H \ F	0—7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28—35
AS1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
AS2	1	3	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
AS3	2	16	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
AS4	3	16	1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
	4																						
	5																						
M1	6	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
M2	7	16	1	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	
M3	8	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	
M4	9	16	1	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	
	10																						
	11																						
D1	12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
D2	13	16	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	
D3	14	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	
D4	15	16	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	
Exit	16																						

Megjegyzés: Valamennyi cím decimális.

4.5. Fortran vagy hasonló algoritmikus nyelv felhasználásával írjunk programot a (4.3) és (4.4) terminálutasításokkal leírt párhuzamos összeadó szimulálására!

4.6. Legyen X és Y a két összeadandó. Mutassuk be a fixpontos szorzási folyamatot lépésről lépésre az egyes regiszterekben a 4.10. ábrán bemutatott módon, a következő esetekre:

a) $X = +101010$ és $Y = -111001$
b) $X = -011101$ és $Y = -011110$
c) $X = +110101$ és $Y = +010111$

4.7. Legyen X és Y az osztandó, ill. az osztó. Mutassuk be a fixpontos osztási folyamatot lépésről-lépésre az egyes regiszterekben, a 4.13. ábrán bemutatott módon, a következő esetekre:

a) $X = -10001010$ és $Y = +1010$
b) $X = +11001111$ és $Y = -1010$
c) $X = +11111111$ és $Y = +1001$

4.8. Algoritmikus nyelv alkalmazásával írjunk programot a következők vizsgálatára:

a) a 4.6. és 4.7. ábrán leírt fixpontos összeadásra és kivonásra;
b) a 4.8. és 4.9. ábrán leírt fixpontos szorzásra;
c) a 4.11. és 4.12. ábrán leírt fixpontos osztásra!

4.9. Módosítsuk a 4.6. és 4.7. folyamatábrákat a fixpontos összeadáshoz és kivonáshoz arra az esetre, ha a kivonás a kivonandó 2-es komplemensének hozzáadásával történik!

4.10. Módosítsuk a 4.6. és 4.7. ábrán levő folyamatábrákat, hogy azok előjeles, 2-es komplemens ábrázolású bináris számok fixpontos összeadására és kivonására alkalmasak legyenek!

4.11. Módosítsuk a 4.8. és 4.9. folyamatábrákat arra az esetre, ha a számábrázolás előjeles, 2-es komplemens ábrázolású, bináris, fixpontos!

4.12. Hogyan módosítsuk a 4.11. és 4.12. folyamatábrákat, ha azok előjeles, 2-es komplemens bináris számok fixpontos osztására szolgálnak?

4.13. Hány végrehajtási ciklus szükséges a (4.12)-ben CDL utasításokkal leírt fixpontos szorzáshoz? Hogyan lehet a szorzási folyamatot rövidíteni?

4.14. Hány végrehajtási ciklus szükséges a (4.13)-ban CDL utasításokkal leírt fixpontos osztáshoz? Hogyan lehet az osztási folyamatot rövidíteni?

4.15. Fortran vagy hasonló algoritmikus nyelv alkalmazásával írjunk programot a (4.37) terminálutasítással leírt párhuzamos összeadó szimulálására!

4.16. Legyenek az AC, TR és MQ regiszterek kétlépcsősek. Írjuk fel ennek megfelelően a (4.12) szorzási sorozatot!

4.17. Ismételjük meg a 4.16. feladatot a (4.13) osztási sorozatra!

4.18. A (4.67) vezérlési struktúrában a vezérlőtárat a főtár első 256 helyével helyettesítjük. Rajzoljuk fel a vezérlési struktúrát és módosítsuk, ha szükséges az időzítést és a vezérlőjeleket! Határozzuk meg a vezérlésre használt utasítás formátumát, írjuk le a műveletet, az utasítás sorozatot és a mikroprogramot!

Irodalom

[1] BURKS, S.W.—GOLDSTINE, H. H.—von NEUMANN, J.: *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument.* 1946, Datamation 8, No. 9, pp. 24...31, September, 1962, and 8, No. 10, pp. 36...41, October, 1962.

[2] PHISTER, M., JR.: *Logical Design of Digital Computers.* New York, John Wiley & Sons, Inc., 1958.

[3] BLAAUW, G. A.: *Indexing and Control-Word Technique.* IBM Journal of Resarch and Development 3, No. 3, pp. 288...301, July, 1959.

[4] BECKMANN, F. S.—BROOKS, F. P., JR.—LAWLESS W. JR.: *Developments in the Logical Organization of Computer Arithmetic and Control Units.* Proceedings IRE, 49, No. 1, pp. 53...66, January, 1961.

[5] MacSORLEY,' O. L.: *Hig-Speed Arithmetic in Binary Computers.* Proceedings IRE, 49, No. 1, pp. 67...101, January, 1961.

[6] CHU, Y.: *Digital Computer Design Fundamentals.* New York, McGraw-Hill Book Company, 1962.

[7] CHU, Y.: *IBM 7094 Principles of Operation.* Form A22—6703—4, 5th edition, October, 21, 1966.

[8] FLORES, I.: *Computer Design.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1967.

[9] GSCHWID, H. W.: *Design of Digital Computers.* Springer-Verlag, New York, Inc., 1967.

[10] CHU, Y.: *Introduction to Computer Organization.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1970.

5.
Párhuzamos, bináris, lebegőpontos aritmetikai egység

Ha a számítás folyamán előforduló számok értéktartománya nagyon nagy vagy nagyon kicsi lesz, a programozó számára igen nehéz és időt rabló munka, hogy állandóan figyelemmel kísérje az egész számítás alatt valamennyi szám bináris (vagy decimális) pontját. Ezért kívánatos, hogy a számokat hardware-ben, exponenciális alakban reprezentáljuk. Az ilyen alakban ábrázolt számokat *lebegőpontos számoknak* nevezzük. A lebegőpontos számok összeadását, kivonását, szorzását és osztását kezelő aritmetikai egységet *lebegőpontos aritmetikai egységnek* nevezzük. A lebegőpontos aritmetika megvalósítása felszabadítja a programozót a számok helyérték-változtatásának terhétől a számítás alatt.

Ez a fejezet olyan párhuzamos, bináris aritmetikai egységet ír le, amely lebegőpontos összeadást, kivonást, szorzást és osztást hajt végre. Az előzőkben tárgyalt fixpontos aritmetikai egységgel összehasonlítva a lebegőpontos aritmetikai egység összetettebb, mivel realizálásához több funkció szükséges.

5.1.
Az aritmetikai egység struktúrája

Az aritmetikai egység leírását a lebegőpontos számábrázolással, a struktúrával és a párhuzamos összeadóval kezdjük. Azokról a funkcionális elemekről, amelyek a lebegőpontos összeadás, kivonás, szorzás és osztás lebonyolításához szükségesek, a következő pontban lesz szó.

5.1.1.
Lebegőpontos számábrázolás

Egy N számot exponenciális alakban a következőképpen ábrázolhatunk.

$$N = f \cdot r^{e+b}, \tag{5.1}$$

ahol r a számrendszer alapja (radix); b eltolás; e kitevő; f előjeles szám.
A fenti r alapszám, b eltolás és e kitevő egész szám, míg az f a legtöbb esetben előjeles törtszám.

Adott szóhossz esetén a lebegőpontos ábrázolás megnöveli a számábrázolás tartományát, de csökkenti a szám értékes bitjeinek számát. Továbbá, amint alábbiakban látni fogjuk, a lebegőpontos összeadáshoz vagy kivonáshoz szükséges idő rendszerint hosszabb, mint a fixpontos összeadás vagy kivonás ideje, viszont a lebegőpontos szorzás vagy osztás felhasznált ideje rövidebb lehet, mint a fixpontos szorzásé vagy osztásé.

A lebegőpontos aritmetikai egység leírásához a következő értékeket és számokat választottuk:

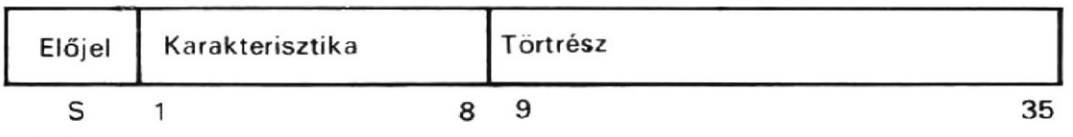
$$\begin{aligned} r &= 2, \\ e &= \text{8-bites, előjeles egész szám,} \\ b &= 200_8, \\ f &= \text{28-bites, előjeles törtszám} \\ &\quad [\text{azaz } |f| < 1]. \end{aligned} \tag{5.2}$$

Továbbiakban, az $(e+b)$ mennyiséget a lebegőpontos szám karakterisztikájának nevezzük, vagyis:

$$k = e+b. \tag{5.3}$$

Mivel az r alapszám és a b eltolás ismert, az egyes lebegőpontos számokat egyszerűen az e kitevővel és az f törtrésszel, vagy pedig a k karakterisztikával és az f törtrésszel lehet ábrázolni.

A lebegőpontos számok ábrázolásához a fentiekben megválasztott értékekkel és számokkal 36 bit szükséges, ebből 8 bit a kitevő és 28 bit az előjeles törtrész. A fentiek alapján a 36-bites lebegőpontos számok formátumát az 5.1. ábrán láthatjuk. Ezen az ábrán a k karakterisztikát és az f törtrészt (az e és f helyett) választottuk a lebegőpontos szám ábrázolására. A karakterisztika 8-bites, előjel nélküli szám, amely az 1-től 8-ig terjedő bitpozíciókon helyezkedik el. Az előjeles törtrész előjellel és abszolút értékkel van ábrázolva, az előjelbit a 0 bitpozícióban, az abszolút érték (a mennyiségrész) pedig a 9—35 bitpozíciókon helyezkedik el. A bináris pont a 8 és a 9 pozíció között van.



5.1. ábra
Lebegőpontos szám formátuma

5.1.2.
A karakterisztika

Az 5.1. ábrán bemutatott formátumú lebegőpontos szám karakterisztikájának lehetséges értékeit az 5.1. táblázat mutatja. Amint a táblázatból látható, a karakterisztikát úgy kapjuk meg, hogy 200_8-at adunk hozzá a kitevő értékéhez. A kitevő 0, ha a karakterisztika 200_8. A kitevő negatív, ha a karakterisztika $0—177_8$. Tehát a karakterisztika értékei a minimális 0-tól 377_8-ig terjednek.

5.1. táblázat

A karakterisztika értékei

Karakterisztika (oktális)	Kitevő (decimális)
alulcsordulás	kisebb, mint (-128_{10})
000	-128_{10}
001_8	-127_{10}
177_8	-1
200_8	0
201_8	$+1$
377_8	$+127_{10}$
túlcsordulás	nagyobb, mint $(+127_{10})$

Ha a karakterisztika értéke nagyobb 377_8-nál, a szám túllépi azt a felső karakterisztikakorlátot, amelyet az adott formátumban ábrázolni lehet; ezt az állapotot *lebegőpontos túlcsordulásnak* nevezzük. Az a legnagyobb pozitív szám, amelyet ily módon ábrázolni tudunk:

$$.77777777_8 \times 2^{177_8+200_8}$$

Amennyiben a karakterisztika kisebb 0-nál, a szám túllépi az alsó karakterisztikakorlátot; ezt az állapotot *lebegőpontos alulcsordulásnak* nevezzük. Ily módon az ábrázolható legkisebb pozitív szám:

$$.000000001_8 \times 2^{-200_8+200_8}$$

5.1.3.
Törtrész

Egy lebegőpontos számot sokféle különböző alakban lehet ábrázolni, mindegyiket más törtrésszel. Azt a speciális lebegőpontos számábrázolást nevezzük *normál* (vagy *normalizált*) *alaknak*, amikor a törtrész legértékesebb számjegye nem nulla. Így a törtrész abszolút értéke kisebb 1-nél, de egyenlő vagy nagyobb, mint $1/r$. Az 5.1. ábrán látható ábrázolási mód esetén a normál alakhoz az szükséges, hogy a 9-edik bitpozícióban 1 legyen, és így a törtrész abszolút értéke

$$\frac{1}{2} \leqq |f| < 1. \qquad (5.4)$$

A lebegőpontos szám normál alakja tehát az egyik lehetséges szám egyedi ábrázolási mód. A nemnormál alakból normál alakba való átalakítást *normalizálásnak* nevezzük. A nulla törtrészű lebegőpontos számot nem lehet normalizálni.

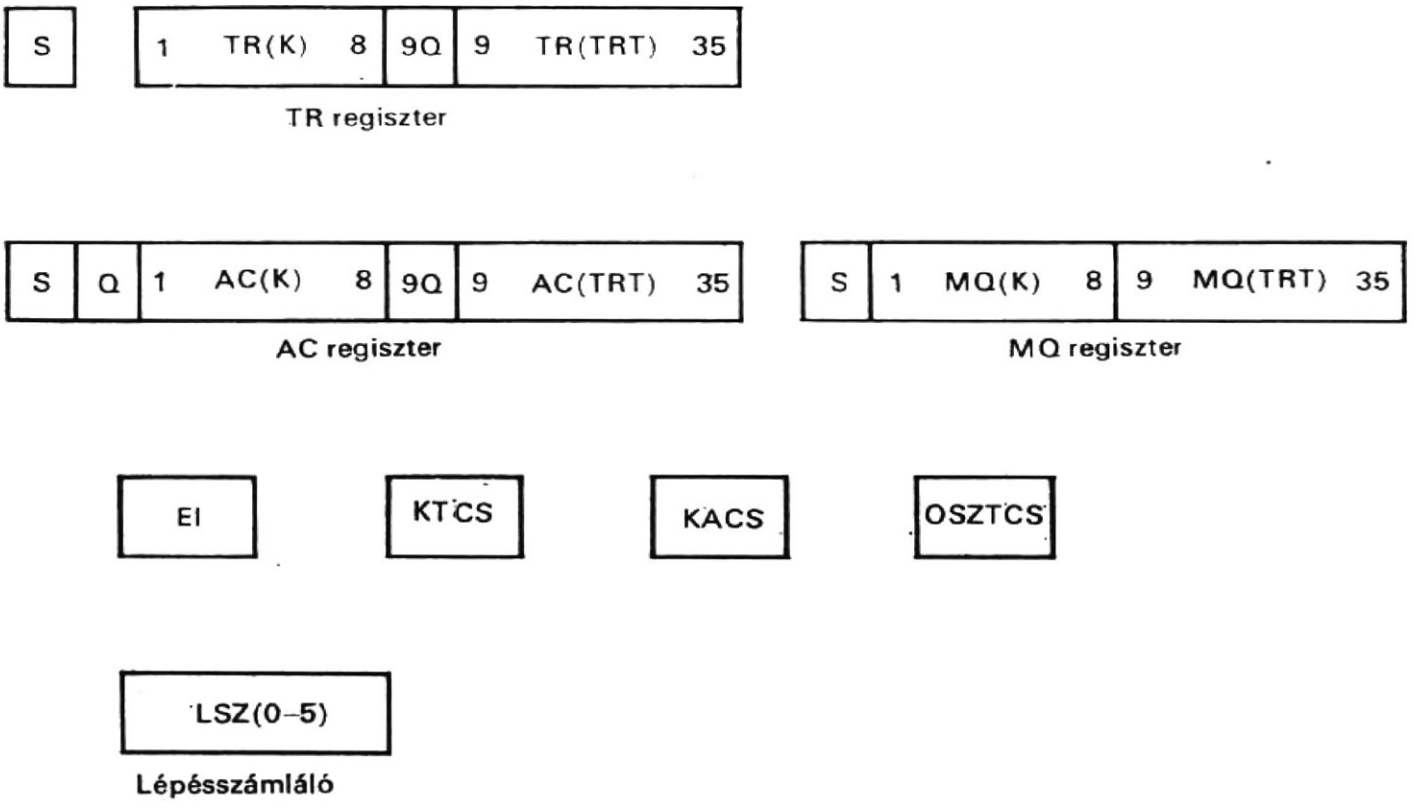
Megállapodás alapján *normál nullának* definiáljuk azt a lebegőpontos számot, amelynek mind a karakterisztikája, mind a törtrésze nulla.

5.1.4. Struktúra

A párhuzamos, bináris, lebegőpontos aritmetikai egység három regisztert tartalmaz: AC, MQ és TR, valamint négy darab egybites indikátort, amint az 5.2. ábrán láthatjuk. A három regiszter mindegyike két részre oszlik: az egyik rész a lebegőpontos szám karakterisztikáját, a másik pedig a törtrészét tartalmazza. Pontosabban a TR(K), AC(K) és MQ(K) regiszterek karakterisztikát ábrázoló részei, míg a TR(TRT), AC(TRT) és MQ(TRT) a törtrészhez tartozó részek. Az EI regiszter tárolja az eredmény előjelét. A KTCS és KACS regiszterek jelzik a lebegőpontos túlcsordulást, ill. alulcsordulást. Az OSZTCS regiszter az osztási túlcsordulást indikálja.

Az alábbiakban CDL utasításokkal leírjuk az 5.2. ábrán látható struktúrát.

```
Comment, a párhuzamos, lebegőpontos aritmetikai egység struktúrája      (5.5)
Register,   AC(S, Q, 1—8, Q, 9—35),      $akkumulátor
            TR(S, 1—35),          e      $ tár tartalomregisztere
            MQ(S, 1—35),                 $szorzó—hányados regiszter
            LSZ(0—5),                    $lépésszámláló
            EI,                          $előjelindikátor
            KTCS,                        $lebegőpontos túlcsordulás-
                                          indikátor
```



5.2. ábra
Párhuzamos, bináris, lebegőpontos aritmetikai egység konfigurációja

```
                KACS,                    $lebegőpontos alulcsordulás-
                                         indikátor
                OSZTCS,                  $osztási túlcsordulás-
                                         indikátor
Subregister,    AC(K)=AC(1—8),           $az AC karakterisztika része
                TR(K)=TR(1—8),           $a TR karakterisztika része
                MQ(K)=MQ(1—8),           $az MQ karakterisztika része
                AC(TRT)=AC(9—35),        $az AC törtrésze
                TR(TRT)=TR(9—35),        $a TR törtrésze
                MQ(TRT)=MQ(9—35),        $az MQ törtrésze
```

Comment, párhuzamos összeadó a karakterisztika összeadására (5.6)

```
Terminal,   ADD(Q, 1—8)=ADTR(Q, 1—8)⊕ADAC(Q, 1—8)⊕C(Q, 1—8),
            C(Q, 1—7)=ADTR(1—8)*ADAC(1—8)+C(1*8)*ADAC(1—8)+
                      +C(1—8)*ADTR(1—8),
            C(8)=0,
```

Comment, párhuzamos összeadó a törtrész összeadására (5.7)

```
Terminal,   ADD(9Q, 9—35)=ADTR(9Q, 9—35)⊕
                          ADAC(9Q, 9—35)⊕C(9Q, 9—34),
            C(9Q, 9—34)=ADTR(9—35)*ADAC(9—35)+
                          +ADAC(9—35)*C(9—35)+
                      +C(9—35)*ADTR(9—35),
            C(35)=0,
```

Comment, a Z terminal deklarációja (5.8)

```
Terminal,   Z(Q, 1—8)=0—AC(K) add 0—TR(K)
```

Éppúgy, mint a fixpontos aritmetikai egységnél, itt is további bitek: AC(Q) és AC(9Q) szükségesek a lebegőpontos aritmetikai művelet elvégzéséhez. Az AC(Q) bit a karakterisztika legértékesebb bitjéből érkező átvitelbitet tárolja, míg az AC(9Q) bit a törtrész legértékesebb bitjéből származó bit tárolására szolgál. Ezeket a biteket általában az összeadás vagy a kivonás után megvizsgálják, annak meghatározására, előfordult-e túlcsordulás.

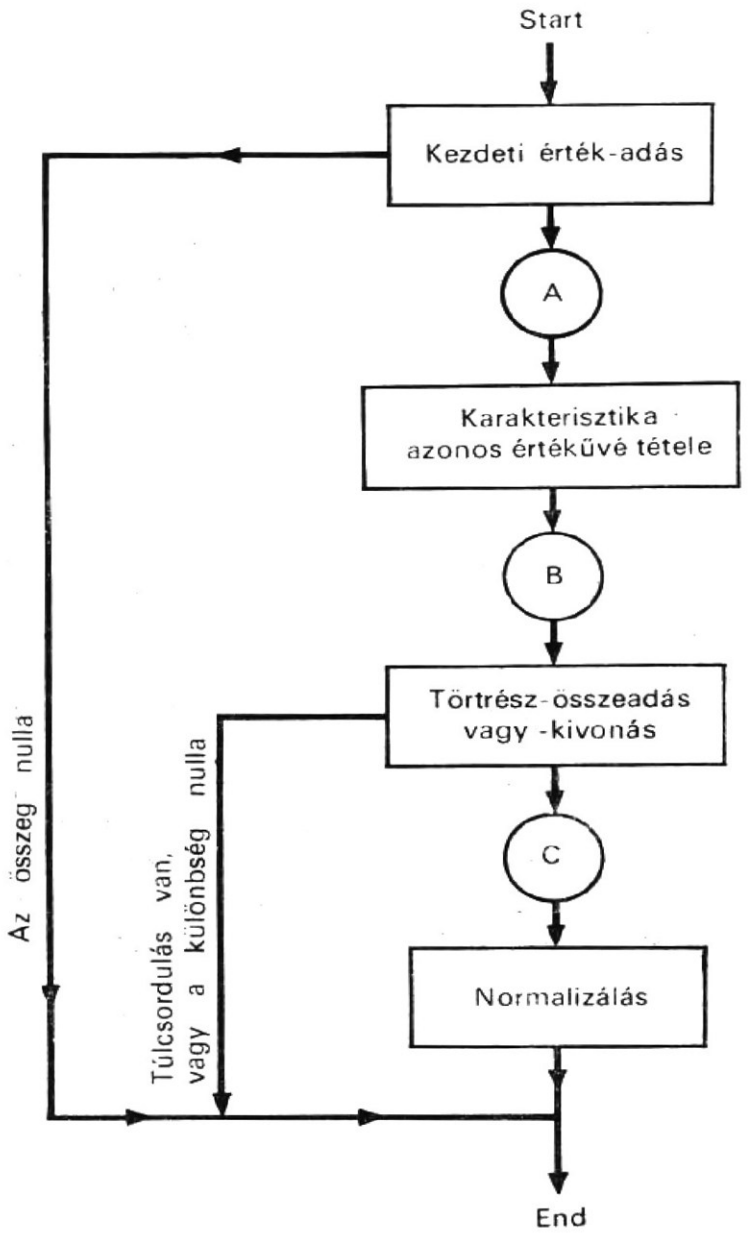
Két párhuzamos összeadót használunk, egyet a karakterisztika összeadására és a másikat a törtrész összeadására. Ezt a két összeadót az (5.6) és (5.7) terminálutasítások írják le, ahol ADTR(Q, 1 — 8) és ADAC(Q, 1 — 8) a bemeneti kapocspárok, C(Q, 1 — 8) az átvitelek és ADD(Q, 1—8) a kimeneti kapocspárok a karakterisztika összeadására szolgáló párhuzamos összeadóban. Az ADTR(9Q, 9—35) és ADAC(9Q, 9—35) a bemeneti kapocspárok, a C(9Q, 9—34) az átvitelek és az ADD(9Q, 9—35) a kimeneti kapocspárok a törtrészek összeadását végző párhuzamos összeadóban. A fenti két párhuzamos összeadót kényelmesebb módon is le lehet írni, az add bázisoperátor segítségével.

A következőkben, a lebegőpontos aritmetikai folyamatok leírásakor szükség van egy vizsgálatra (tesztre) is, amely a karakterisztika összeadását végző párhuzamos összeadó egyik kimeneti kapcsát ellenőrzi. A vizsgált párhuzamos összeadó egyik bemenete az AC(K) részregiszter, míg a másik bemenete a TR(K) részregiszter. A párhuzamos összeadó kimeneti kapcsait tekintsük Z-nek. Ezt az (5.8) terminálutasítás deklarálja.

5.2.
Lebegőpontos összeadás és kivonás

A lebegőpontos összeadás a TR regiszterben levő lebegőpontos összeadandót hozzáadja az AC regiszterben elhelyezkedő másik lebegőpontos összeadandóhoz. A lebegőpontos kivonás kivonja a TR regiszterben levő lebegőpontos kivonandót az AC regiszterben levő lebegőpontos kisebbítendőből. Az AC és TR regiszterekben kezdetben elhelyezkedő lebegőpontos operandusokról feltételezzük, hogy normál (normalizált) alakban vannak. Emlékezzünk vissza, hogy ezeknek a lebegőpontos számoknak a törtrészét előjellel és abszolút értékkel ábrázoljuk.

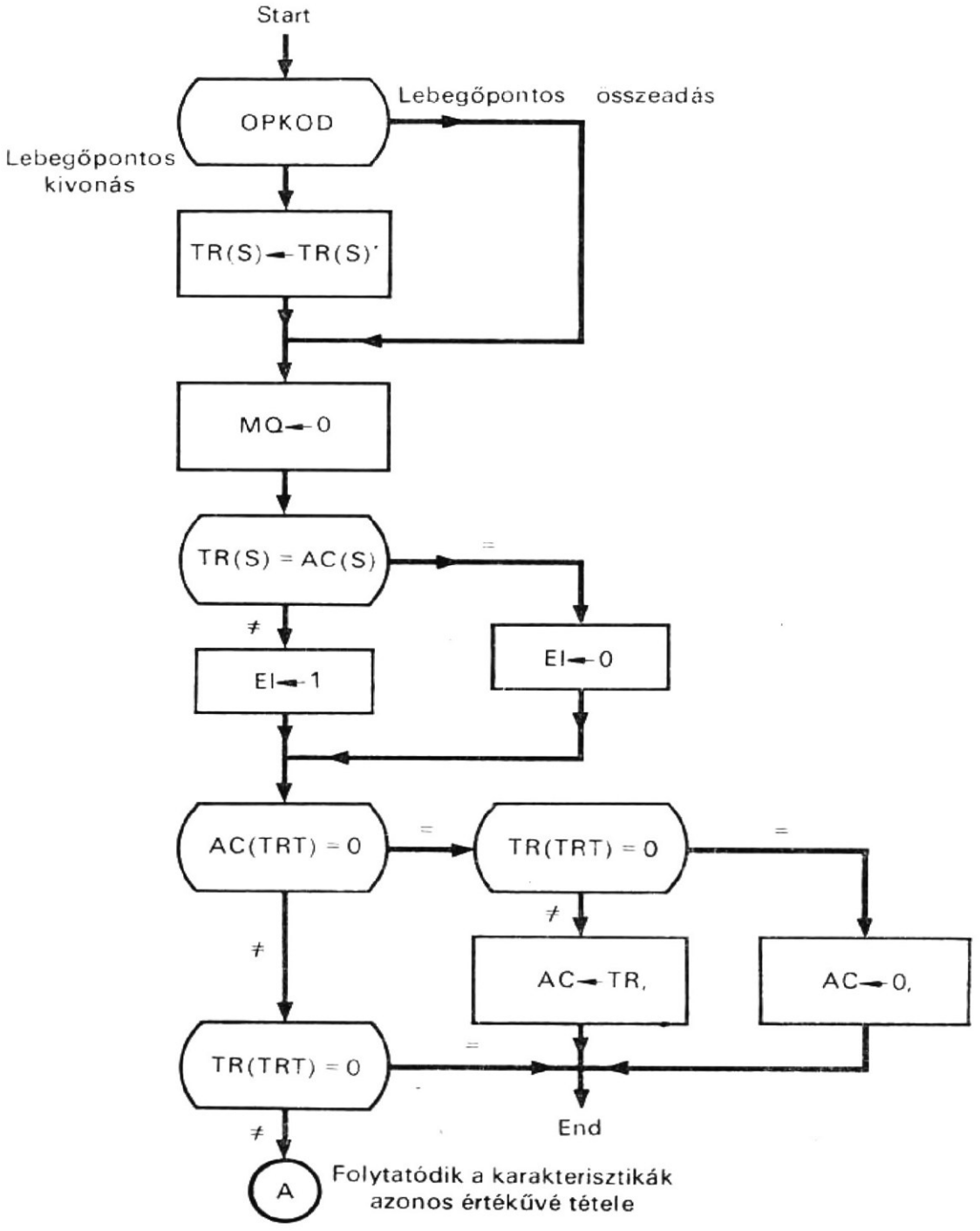
Az 5.3 folyamatábra a lebegőpontos összeadást és kivonást ábrázolja. A kezdetiértékadás után a lebegőpontos összeadáshoz és kivonáshoz először a lebegőpontos számok karakterisztikáit azonos értékűvé tesszük, azután következik a lebegőpontos számok törtrészeinek összeadása vagy kivonása, és végül az összeadás után az összegnek, a kivonás után a különbségnek a normalizálása. Az összeadás közben a lebegőpontos túlcsordulást, a normalizáláskor a lebegőpontos alulcsordulást kijelzik, ha ilyen egyáltalán előfordul.



5.3. ábra
Lebegőpontos összeadás és kivonás

5.2.1. Kezdetiérték-adás

A lebegőpontos összeadás és kivonás kezdetiérték-adó részét az 5.4. folyamatábra szemlélteti. Ennek négy feladata van. A TR regiszterben levő kivonandó előjelét komplementálja kivonás esetén, és így a kivonás összeadássá válhat. Az MQ regiszternek 0 értéket ad, mivel az tartalmazza az AC regiszterben levő törtrész azon hányadát, amelyet az MQ regiszterben helyeznek el a karakterisztikák azonos értékűvé tételével. Összehasonlítja a két összeadandó előjelét. Ha az előjelek megegyeznek, az EI indikátor 0 értéket kap; egyébként pedig értéke 1 lesz. Végül meghatározza az összeget, ha az összeadandók egyikének vagy mindkettőnek nulla a törtrésze. Ha mindkét törtrész nulla, az AC regiszter 0 értéket kap, és így az összeg a normál nulla alakjában van. Ha



5.4. ábra
Folyamatábra a lebegőpontos összeadás és kivonás kezdetiérték-adó részéhez

csak a TR regiszterben levő összeadandó törtrésze nulla, az AC regiszterben levő lebegőpontos összeadandó lesz az összeg és a művelet befejeződött. Ha csak AC regiszterben levő összeadandó törtrésze nulla, a TR regiszterben levő lebegőpontos összeadandó lesz az összeg; tehát az összeadandót át kell vinni a TR regiszterből az AC regiszterbe. Ha egyik törtrész sem nulla, a lebegőpontos összeadás a karakterisztikák azonos értékűvé tételével folytatódik.

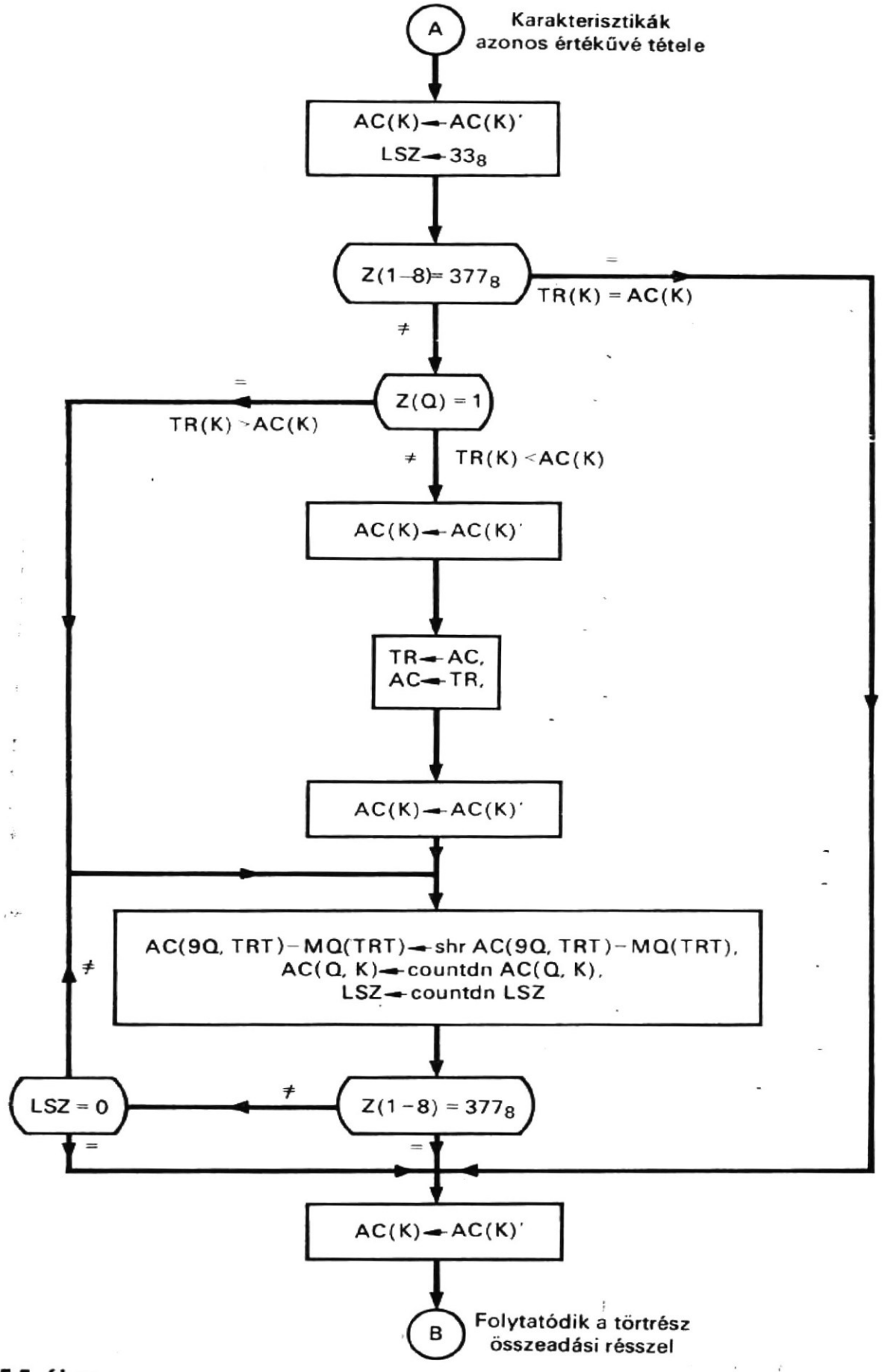
5.2.2. Karakterisztikák azonos értékűvé tétele

Két lebegőpontos szám összeadásakor össze kell adni a lebegőpontos számok törtrészeit, ha azonosak a karakterisztikái. Ha ezeknek a lebegőpontos számoknak nem azonos a karakterisztikájuk, akkor azokat először azonos értékűvé kell tenni, mielőtt az összeadást elvégeznénk. A lebegőpontos összeadás és kivonás karakterisztikák azonos értékűvé tétele látható az 5.5. ábrán.

Két lebegőpontos szám karakterisztikáinak azonos értékűvé tételekor megvan a választásunk, hogy vagy a kisebb karakterisztikájú lebegőpontos szám törtrészét léptetjük jobbra, hogy a karakterisztikájának értékét meg lehessen növelni, vagy a nagyobb karakterisztikájú lebegőpontos szám törtrészét balra léptetjük, hogy karakterisztikájának értékét csökkenteni lehessen. Mivel az MQ regiszter rendelkezésre áll az eltolt bitek tárolására (amit egyébként szorzáskor használnak), az előbbi megoldást választjuk. Minthogy az AC regiszter helyiérték-eltolásra alkalmas léptető-regiszter (ez a szorzáshoz is szükséges), a kisebb karakterisztikájú lebegőpontos számot az AC regiszterben kell elhelyezni. Ezért a karakterisztikák azonos értékűvé tétele a két karakterisztika összehasonlításával kezdődik. Ha ezek egyenlők, nincs szükség az azonos értékűvé tételre. Ha az AC regiszterben levő karakterisztika kisebb, mint a TR regiszterben levő, akkor az AC regiszterben elhelyezkedő lebegőpontos számmal a folyamat elkezdhető. Egyébként pedig meg kell cserélni az AC és a TR regiszterben levő lebegőpontos számokat. Az azonos értékűvé tétel során az AC(9Q) bit és az AC és MQ regiszterek törtrészeinek felfűzéséből létrejött regisztersort egy bitpozícióval jobbra kell léptetni, és ugyanakkor az AC regiszter karakterisztikarészét eggyel meg kell növelni.

A karakterisztikák fenti összehasonlítását kivonás segítségével végzik. Ez a kivonás helyettesíthető a kivonandó 1-es komplemensének összeadásával. Ezért az AC(Q, K) részregiszternek a vizsgálat előtt képezik az 1-es komplemensét és az 1-es komplemens alakban is marad a soron következő kivonási vizsgálatok során mindaddig, amíg a karakterisztikák azonos értékűvé nem válnak, ekkor vissza kell állítani eredeti értékére. Jegyezzük meg, hogy az AC(Q, K) részregiszter 1-es komplemenseinek képzésére az AC és TR regiszterek megcserélése előtt és után van szükség.

Öt mikroművelet vesz részt a karakterisztikák azonos értékűvé tételében. Az AC(9Q, TRT)—MQ(TRT) regisztersor egy bitpozícióval jobbra lép, és ugyanakkor mind az AC(Q, K) részregiszterben levő karakterisztika, mind az LSZ lépésszámláló értéke eggyel csökken. Az AC(Q, K) részregiszter értékét eggyel csökkentjük, ahelyett, hogy növelnék, mivel a részregiszter tartalma 1-es komplemens kódban van. Az AC(Q, K) részregisztert nullára megvizsgálják, és így a folyamat befejeződik, amikor a karakterisztikák azonos értékűvé váltak. Továbbá ellenőrizzük az LSZ lépésszámlálót nullára, és a folyamat akkor is befejeződik, ha a jobbra lépések száma eléri a maximális 27-et.



5.5. ábra
Folyamatábra a lebegőpontos összeadás és kivonás karakterisztika-összeadási részéhez

5.2.3.
Törtrész-összeadás és -kivonás

Két lebegőpontos szám törtrészének összeadását vagy kivonását ábrázolja az 5.6. folyamatábra. Ez a műveletsorozat hasonló a fixpontos számokéhoz. Amint az ábrából látható, az összeadás vagy kivonás szerinti elágazás az EI regiszter tartalma alapján történik. Ha az EI értéke 0, a két operandus előjele megegyezik és összeadásra van szükség; egyébként kivonást kell végrehajtani.

Ha összeadás szükséges, a TR regiszterben levő törtrészt kell hozzáadni az AC regiszterben levő törtrészhez. Megvizsgálják az AC(9Q) bitet. Ha ez 1, az AC(9Q, TRT)—MQ(TRT) sorregisztert egy bitpozícióval jobbra léptetik, hogy az összeg törtszám maradjon. (Ez a lépés egyszerűsíti a soron következő normalizálást, mivel csak a regisztersor balra léptetésére lesz akkor szükség.) Ugyanakkor megnövelik az AC(Q, K) részregiszterben levő karakterisztikát is. Ezután lebegőpontos túlcsordulás vizsgálat következik. Ha túlcsordulás következett be, kijelzik. Akár van túlcsordulás, akár nincs, a lebegőpontos művelet befejeződött.

Ha kivonásra van szükség, az AC regiszterben levő törtrész 1-es komplemensét képezik és ily módon a kivonást a kivonandó 1-es komplemensével végzett összeadásakor helyettesítik. Ezután a TR regiszterben levő törtrészt hozzáadják az AC regiszterben levő törtrészhez. A következőkben megvizsgálják az AC(9Q) bitet. Ha az 1, azt mutatja, hogy a TR regiszterben a törtrész nagyobb, mint az AC regiszterben levő törtrész. Ebben az esetben a ciklikus átvitelt hozzáadják az AC(9Q, TRT) részregiszterben levő törtrészhez és az AC(S) előjelet a TR(S) előjellel helyettesítik, minthogy a különbség a nagyobb operandus előjelét veszi fel. Ha az AC(9Q) bit 0, ez azt mutatja, hogy a TR regiszterben a törtrész egyenlő vagy kisebb, mint az AC regiszterben levő törtrész. Most megvizsgálják a törtrészt az AC regiszterben, vajon csupa 1-est tartalmaz-e. Ha csupa 1-est tartalmaz, a különbség nulla; ennek eredményeképpen az AC regiszter 0 értéket kap. Ha nem csupa 1-est tartalmaz, a különbség negatív; az AC regiszterben a törtrésznek ismét képezik az 1-es komplemensét, hogy a különbség abszolút értékét előállítsák.

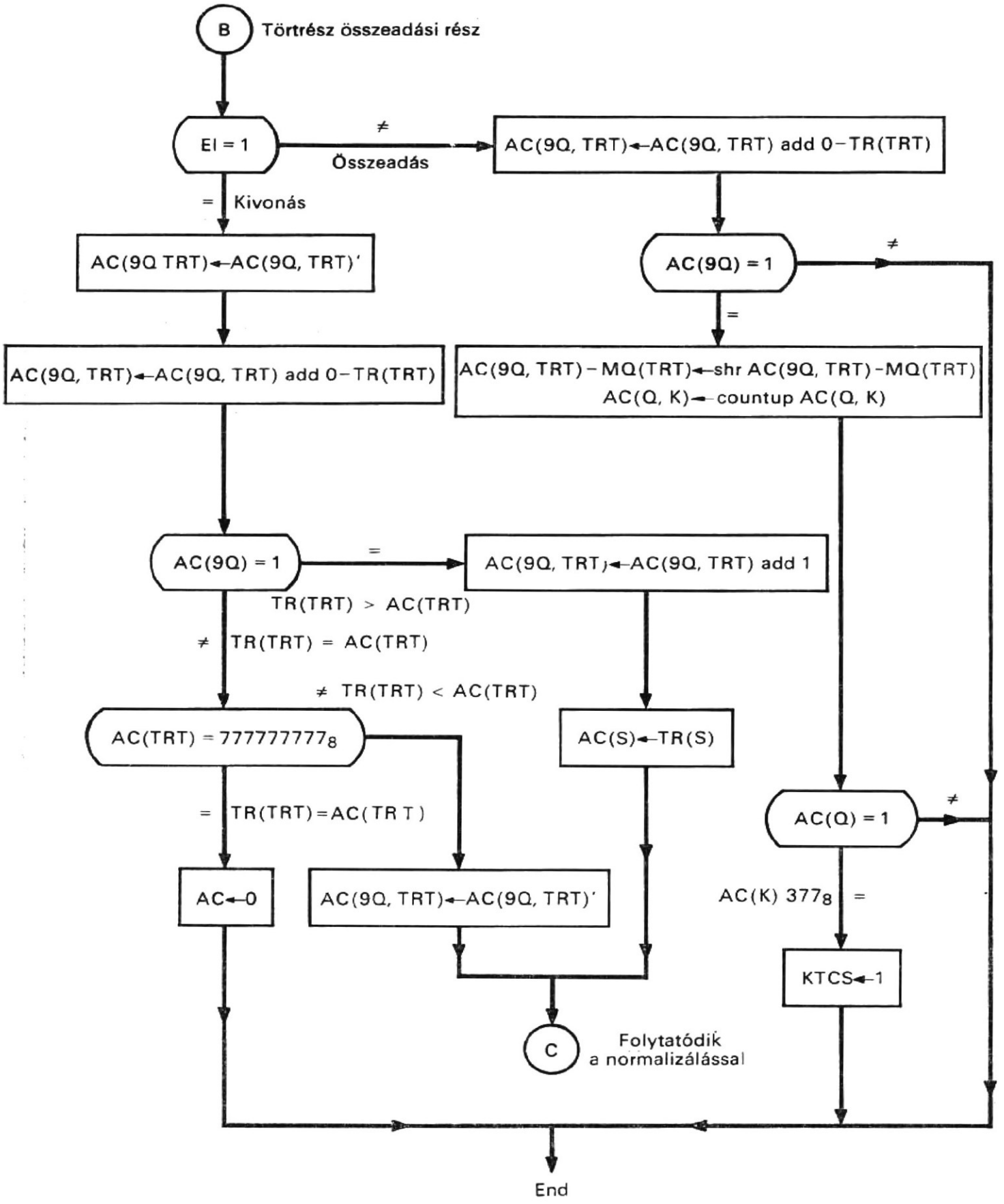
Az összeadás vagy kivonás végén az eredmény értékesebb része az AC(S,TRT) részregiszterben lesz, míg kevésbé értékes része az MQ(TRT) részregiszterben. Ha az összeadás után az összegben nem keletkezett túlcsordulás vagy a kivonás után a különbség nem normál nulla, a sorozat a normalizálással folytatódik.

5.2.4.
Normalizálás

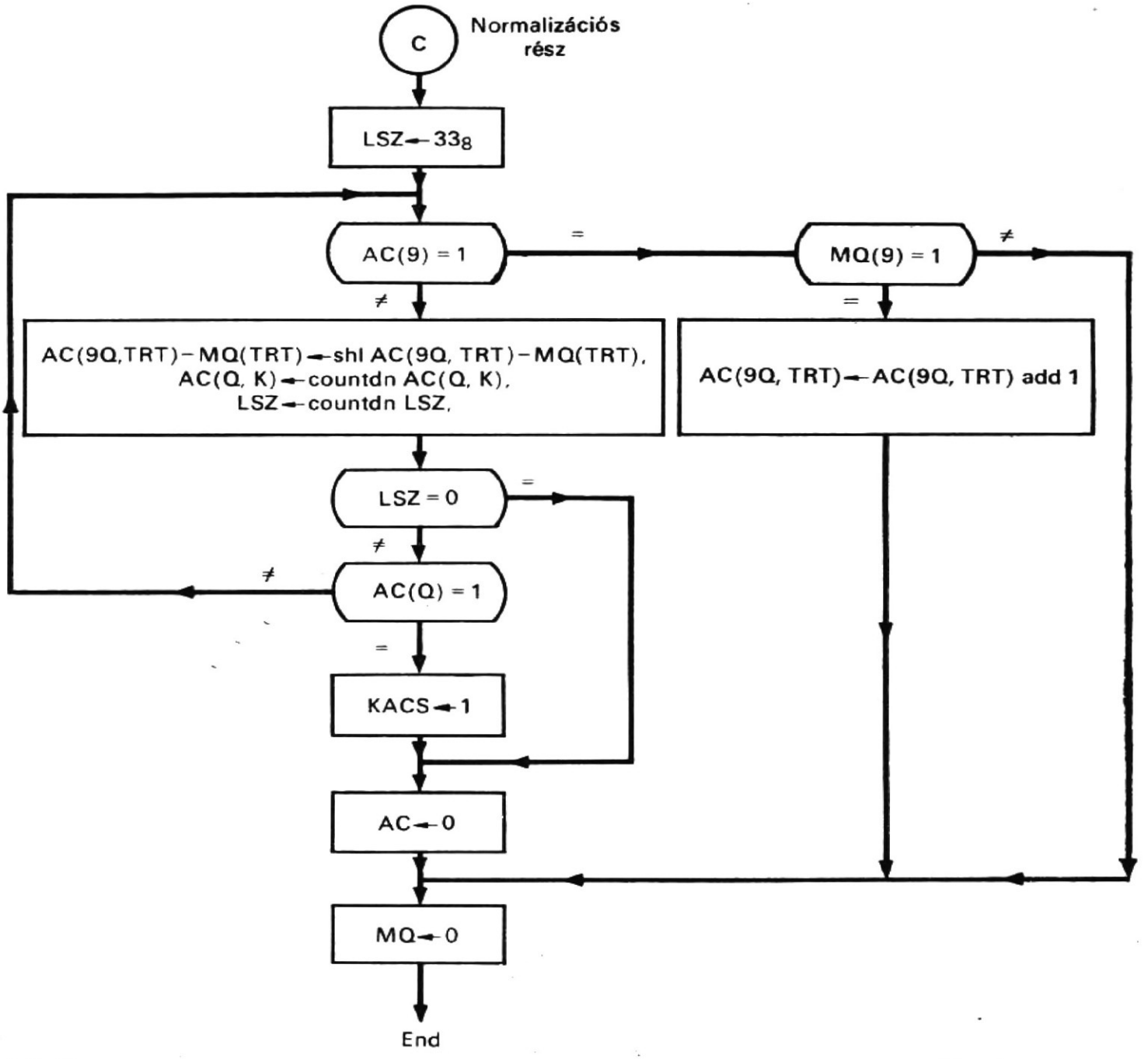
Amint az előzőkben említettük, a lebegőpontos szám normál alakra hozatalának folyamatát *normalizálásnak* nevezzük. A lebegőpontos összeadás és kivonás normalizációs részéhez az 5.7. folyamatábra tartozik.

Az 5.1. ábra szerinti formátumú lebegőpontos szám normalizálásához az kell, hogy a 9-dik bitpozíción 1 álljon. Ezért megvizsgálják az AC(9) bitet. Ha az 1-est tartalmaz, a szám normál alakban van. Ha a bit 0, normalizálásra van szükség.

A normalizálási folyamat azzal kezdődik, hogy a lépésszámláló 27-es értéket kap. Ezután az AC(9Q, TRT)—MQ(TRT) regisztersort 1 bittel balra léptetik és, ugyanakkor 1-gyel csökkentik mind az AC(Q, K) részregiszter, mind az LSZ lépésszámláló tartalmát. Ezután az LSZ lépésszámlálót megvizsgálják nullára. Ha 0-t tartalmaz, a lebegőpontos szám 0, és mind az AC, mind az MQ regiszter 0 értéket kap. Egyébként meg-



5.6. ábra
Folyamatábra a lebegőpontos összeadás és kivonás törtrész-összeadási részéhez



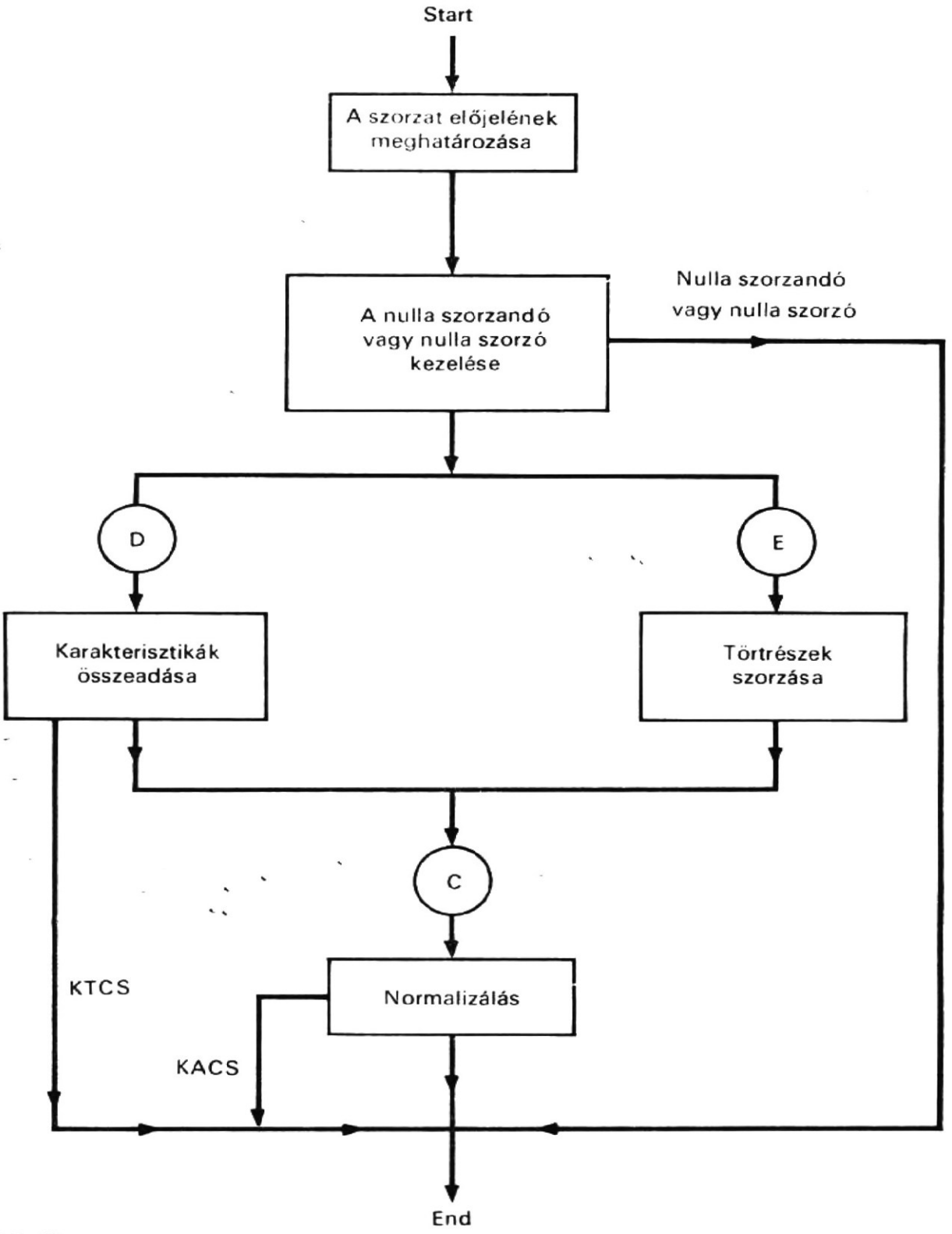
5.7. ábra
Folyamatábra a lebegőpontos összeadás és kivonás normalizációs részéhez

vizsgálják az AC(Q) bitet az esetleges lebegőpontos alulcsordulásra. Ha alulcsordulás van, a KACS regiszter 1 értéket kap; mind az AC, mind MQ értéke 0 lesz, és ez a folyamat befejeződik. Ha sem az LSZ lépésszámláló nem tartalmaz nullát, sem alulcsordulás nem volt, akkor ismét megvizsgálják az AC(9) bitet. A regisztersort újból balra léptetik. Ismét eggyel csökkentik mind az AC(Q, K) részregiszter, mind az LSZ lépésszámláló tartalmát. Újból megvizsgálják az LSZ lépésszámlálót és az AC(Q) bitet 0-ra, ill. 1-re. Ezeket a lépéseket addig ismétlik, amíg 1 nem jelenik meg az AC(9) pozícióban. Ekkor az összeg normál alakban van. Az AC(TRT) részregiszterben a törtrészt most kerekítik oly módon, hogy 1-et adnak az AC(9Q, TRT) részregiszterhez, ha MQ(9)=1, és azután az MQ regiszter 0 értéket kap. Ezzel most már befejeződött a lebegőpontos összeadás és kivonás.

A fenti normalizálásban csak a regisztersor balra léptetésére van szükség, hiszen az AC(9Q)-értéke nem 1, mivel egybites jobbra léptetést végeztünk a normalizálás előtt a törtrész összeadás alatt, amint az 5.6. ábrán látható.

5.3.
Lebegőpontos szorzás

A lebegőpontos szorzás összeszorozza az MQ regiszterben levő lebegőpontos szorzót a TR regiszterben tárolt lebegőpontos szorzandóval és az AC és MQ regiszterekből alkotott regisztersorban helyezi el a képzett szorzatot, amelynek nagyobb helyiértékű része az AC regiszterben, kisebb helyiértékű része pedig az MQ regiszterben van. A TR és MQ regiszterekben levő lebegőpontos operandusokról feltételezzük, hogy

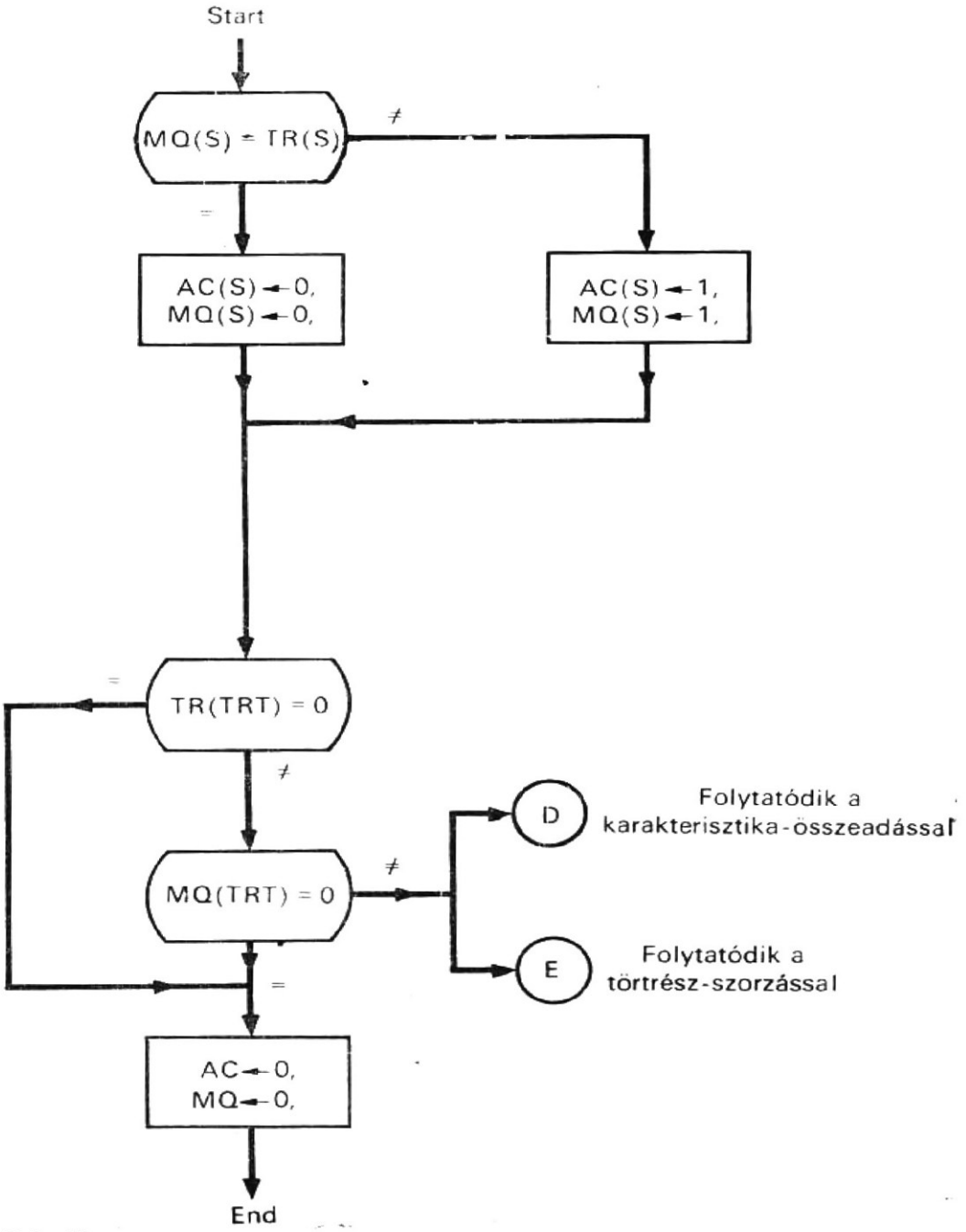


5.8. ábra
Lebegőpontos szorzás

kezdetben normál alakban vannak és törtrészük előjellel és abszolút értékkel van ábrázolva.

Az 5.8. folyamatábrán szemléltetjük a lebegőpontos szorzást. Először meghatározzuk a szorzat előjelét. Minthogy a nulla törtrésze a szorzandóban vagy a szorzóban nulla szorzatot eredményez, megvizsgáljuk a szorzandó és a szorzó nulla törtrészét. Ha nulla törtrészt találunk, a szorzat nulla értéket kap és a szorzási folyamat befejeződött. Egyébként folytatódik a lebegőpontos szorzással.

A lebegőpontos szorzás úgy történik, hogy összeadjuk a TR és az MQ regiszterekben levő karakterisztikarészeket és azután megszorozzuk a TR regiszterben levő törtrészt az MQ regiszterével. Amint az 5.8. ábrán látható, ezek az összeadási és szorzási műveletek szimultán mehetnek végbe. Miután befejeződött a karakterisztikák összeadása és a törtrészek szorzása, a szorzatot normalizáljuk és azután kerekítjük.



5.9. ábra
Folyamatábra a lebegőpontos szorzás kezdetiérték-adó részéhez

5.3.1. Kezdetiérték-adás

A lebegőpontos szorzás kezdetiérték-adását az 5.9. ábrán levő folyamatábrán mutatjuk be. Amint látható, a szorzat előjele 0, ha a szorzandónak és a szorzónak megegyezik az előjele; ha pedig különböző, akkor ez az érték 1. A szorzat előjele mind az AC(S), mind az MQ(S) előjelbitben megtalálható. Ha szorzandónak vagy a szorzónak nulla a törtrésze, a szorzat nulla lesz és az AC és az MQ egyaránt 0 értéket kap; így a szorzat normál nulla lehet. Ha egyik sem nulla, a lebegőpontos szorzás a karakterisztika-összeadással és a törtrész-szorzással folytatódik.

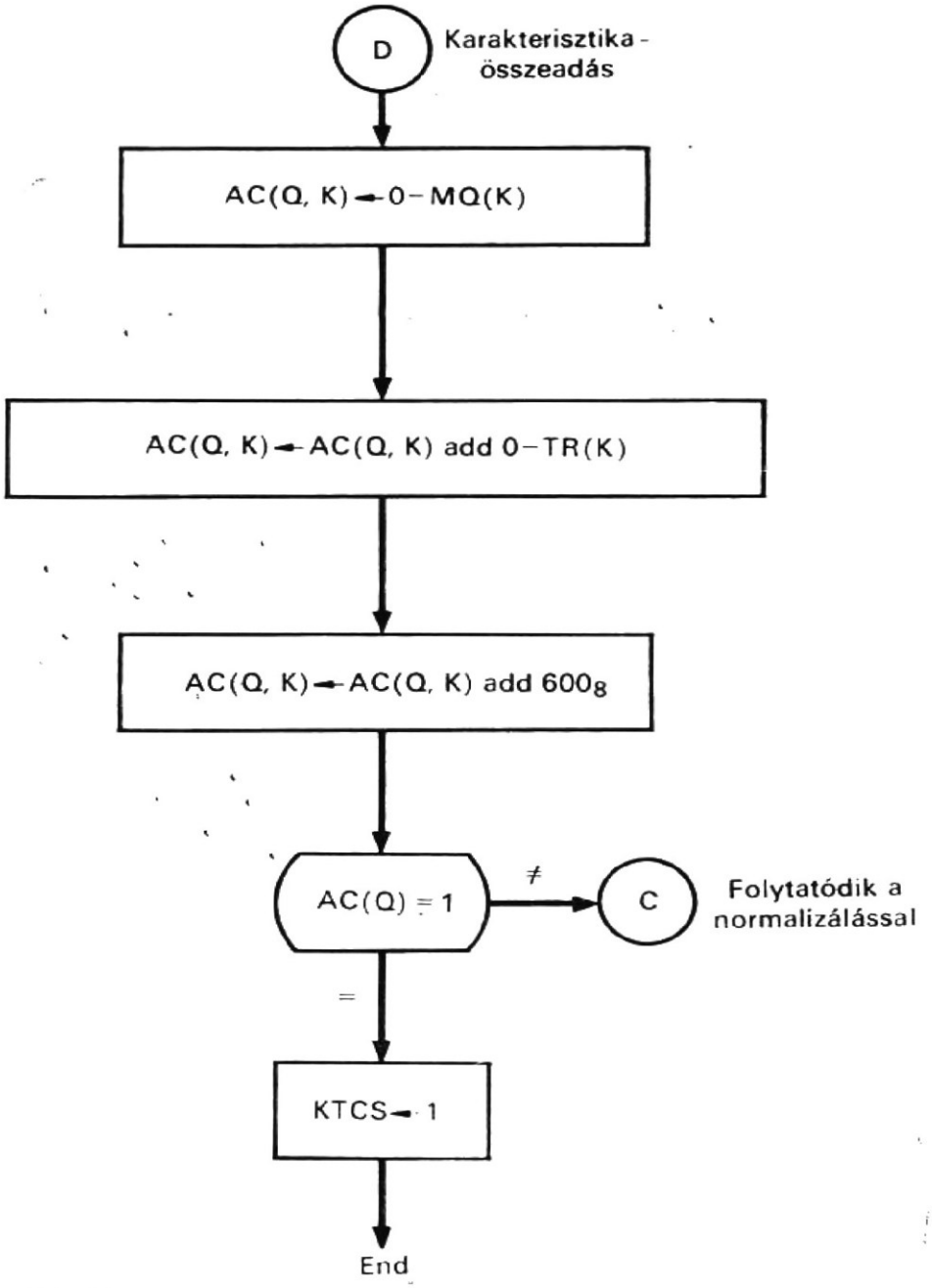
5.3.2. Karakterisztika-összeadás

A karakterisztika-összeadás összeadja a TR(K) részregiszterben levő szorzandó és az MQ(K) részregiszterben elhelyezkedő szorzó karakterisztikáit és a szorzat karakterisztikáját az AC(K) részregiszterben helyezi el. A karakterisztika-összeadás az 5.10. folyamatábrán látható. Az MQ(K) részregiszterben levő szorzót először átviszik az AC(K) részregiszterbe, mivel az összeadás az AC regiszterben megy végbe. Ezután a TR(K) részregiszterben levő szorzandó karakterisztikáját hozzáadják a szorzónak most már, az AC(Q, K) részregiszterben elhelyezkedő karakterisztikájához. Ebből az összegből ki kell vonni a 200_8 mennyiséget, mivel a karakterisztikát úgy kapjuk meg, hogy a kitevőt a 200_8 értékkel eltoljuk. Az AC(K) részregiszterben levő karakterisztika most a két eltolás összegét tartalmazza. A 200_8 kivonása egyenértékű a 2-es komplemensének, azaz 600_8-nak a hozzáadásával. Ezután az AC(Q) bit vizsgálata következik. Ha 1-est tartalmaz, ez azt jelzi, hogy az AC(K) részregiszterben a karakterisztika nagyobb, mint 377_8; tehát lebegőpontos túlcsordulás következett be. Ha túlcsordulás van, a KTCS túlcsordulás-indikátor 1 értéket kap és a lebegőpontosszorzás folyamata befejeződik. Ha az AC(Q) bit 0-át tartalmaz, a lebegőpontos szorzás folyamatát normalizálás követi.

5.3.3. Törtrészek szorzása

A szorzó és a szorzandó karakterisztikáit össze kell adni, a törtrészeiket pedig meg kell egymással szorozni. Az 5.11. folyamatábra a két törtrész szorzását mutatja. Amint az 5.11. ábrából látható, a törtrészek szorzása két fixpontos szám szorzásaként megy végbe; ehhez az ismételt összeadások módszerét használják.

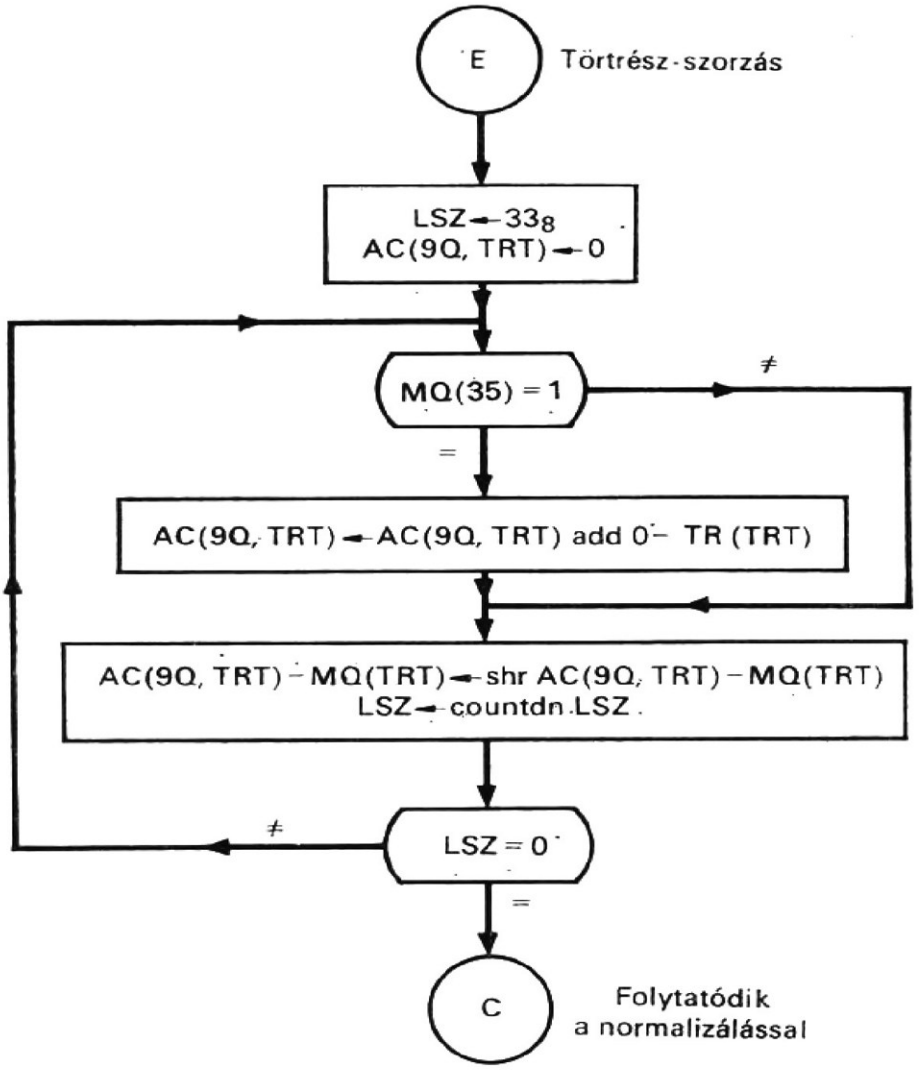
Az 5.11. ábrán az LSZ lépésszámláló kezdetben 27 értéket kap, mivel 27 bit van a törtrészben. Az AC(9Q, TRT) részregiszter kezdetben 0 értéket kap. Ezután megvizsgálják az MQ(35) bitet 1-re. Ha 1-est tartalmaz, a két törtrészt össze kell adni; egyébként nem történik összeadás. Mindkét esetben az AC(9Q, TRT)—MQ(TRT) részregisztert ez után egy bitpozícióval jobbra kell léptetni és ugyanakkor az LSZ lépésszámláló értékét 1-gyel csökkenteni kell. Ezután az LSZ lépésszámlálót 0-ra ellenőrzik. Ha értéke nem nulla, a következő négy lépés: az MQ(35) bit vizsgálata, az összeadás vagy nemösszeadás, a léptetés jobbra és az LSZ lépésszámláló vizsgálata mindaddig folytatódik, amíg értéke 0 nem lesz. Ekkor a lebegőpontos szorzás folyamata a normalizálással folytatódik.



5.10. ábra
Folyamatábra a lebegőpontos szorzás karakterisztika-összeadási részéhez

5.3.4. Normalizálás

A szorzat most az AC(9Q, TRT)—MQ(TRT) regisztersorban van; a nagyobb helyiértékű része az AC(TRT) részregiszterben, míg a kisebb helyiértékű része az MQ(TRT) részregiszterben. A szorzat normalizálását az 5.7. folyamatábrán bemutatott algoritmus felhasználásával lehet elvégezni. Most nem lehetséges normál nulla az AC regiszterben, mivel azt, hogy akár a szorzó, akár a szorzandó törtrésze nulla-e, már a kezdetiérték-adó részben megvizsgálták, amint az az 5.9. folyamatábrán látható.

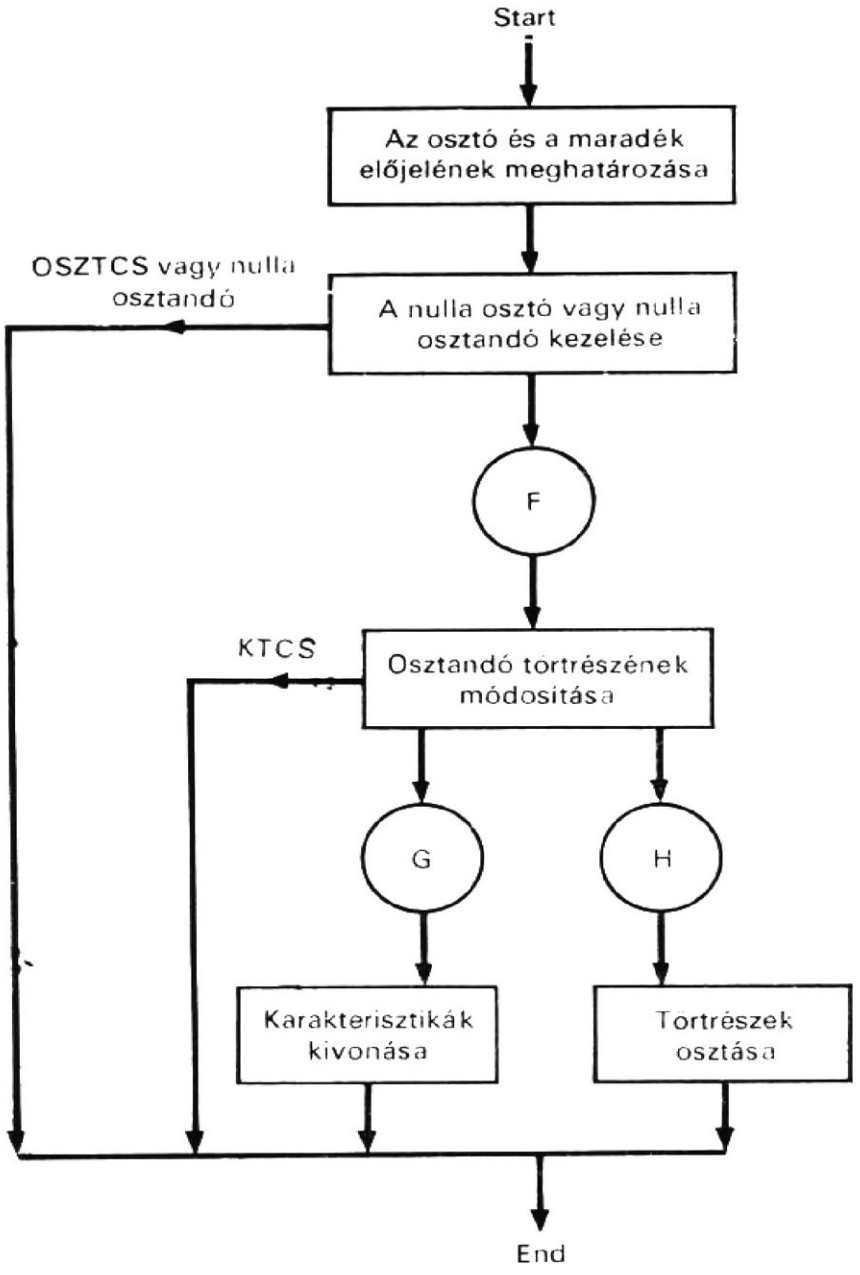


5.11. ábra
Folyamatábra a lebegőpontos szorzás törtrész-szorzási részéhez

5.4. Lebegőpontos osztás

A lebegőpontos osztás elosztja az AC regiszterben levő lebegőpontos osztandót a TR regiszterben elhelyezett lebegőpontos osztóval. A lebegőpontos hányadost az MQ regiszterben, a maradékot az AC regiszterben helyezi el. Az MQ regisztert kezdetben nulla értékre állítják. A lebegőpontos operandusok az AC és TR regiszterekben kezdetben normál alakban vannak, törtrészeik ábrázolása előjellel és abszolút értékkel történik. A hányadost az MQ regiszterben normál alakban állítjuk elő.

Az 5.12. ábra szemlélteti a lebegőpontos osztás egyes lépéseit. Először meghatározzák a hányados és a maradék előjelét. Ha az osztandó nulla, az eredmény is nulla. Ha az osztó nulla, osztási túlcsordulás lép fel. Mindkét esetben az osztási folyamat leáll. Ha egyik eset sem fordul elő, megkezdődik a lebegőpontos osztás. Először az osztandó törtrészének módosítása kezdődik (l. 5.4.2. pontot), amit az osztó karakterisztikájának az osztandó karakterisztikájából való kivonás követ, majd pedig az osztandó törtrészének az osztó törtrészével való osztása megy végbe. Amint az 5.12. ábrán látható, a karakterisztika-kivonás és a törtrész-osztás most is egyidejűleg történhet.

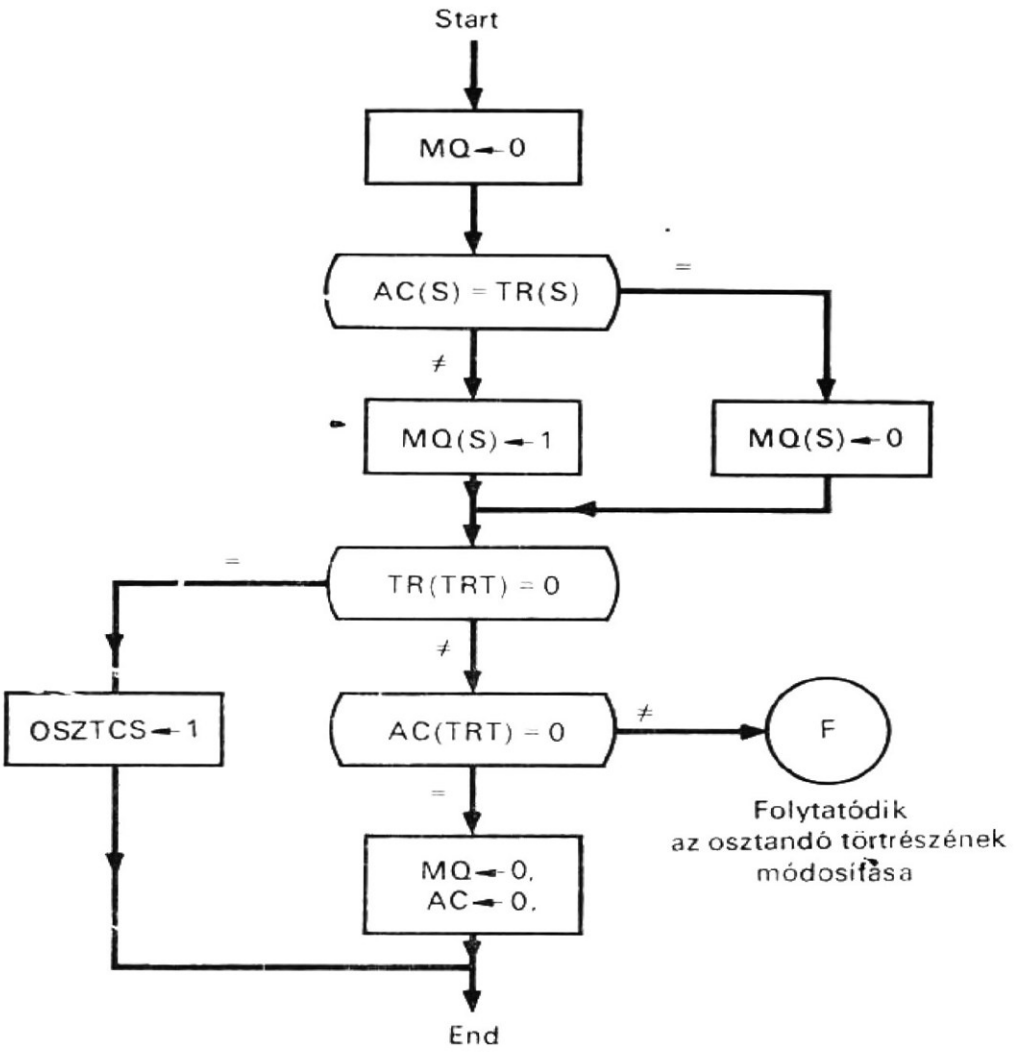


5.12. ábra
Lebegőpontos osztás

5.4.1. Kezdetiérték-adás

A lebegőpontos osztás kezdetiérték-adó részét az 5.13. folyamatábra mutatja. Az MQ regiszter először 0 értéket kap, ezután meghatározzák a hányados előjelét és azt az MQ(S) bitben tárolják. Az AC(S) bit, az osztandó előjele, változatlan marad, ez a maradék előjele.

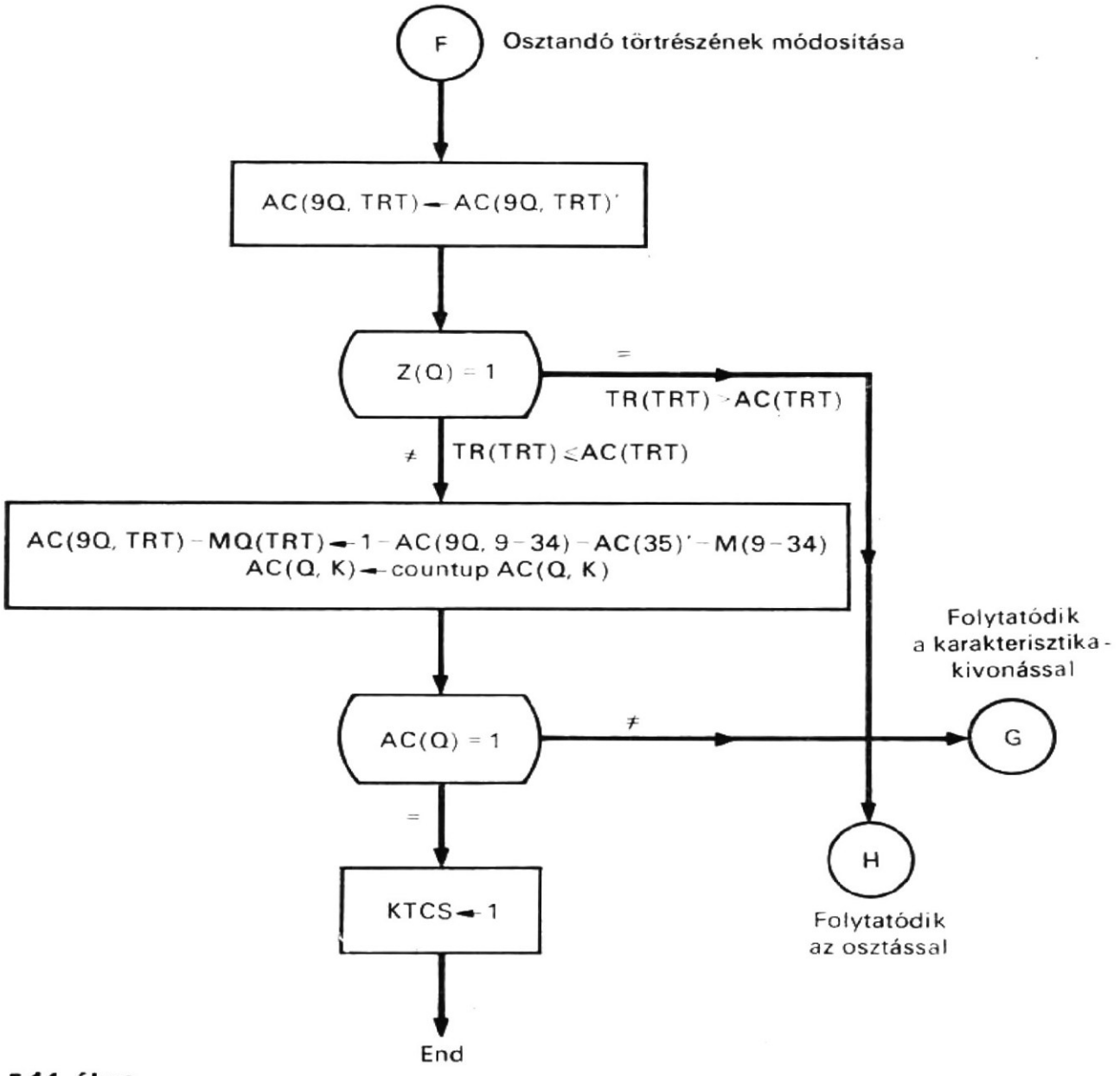
Amint az 5.13. ábrából látható, a TR(TRT) részregisztert 0-ra megvizsgálják. Ha ez nulla, akkor az osztónak nulla a törtrésze; az OSZTCS osztási túlcsordulás-indikátor 1 értéket kap és az osztási folyamat leáll. Ha a TR(TRT) részregiszterben nem nulla a törtrész, megvizsgálják, hogy az AC(TRT) részregiszter értéke nulla-e. Ha nulla, az osztandónak nulla a törtrésze és a hányados nulla lesz; ekkor mind az AC, mind az MQ regiszter 0 értéket kap és az osztási folyamat leáll. Ha sem a TR(TRT), sem az AC(TRT) részregiszter nem nulla, az osztási folyamat az osztandó törtrészének módosításával folytatódik.



5.13. ábra
Folyamatábra a lebegőpontos osztás kezdetiérték-adó részéhez

5.4.2.
Az osztandó törtrészeinek módosítása

Mielőtt két, lebegőpontos szám törtrészének osztása megkezdődnék, az osztandó törtrészét módosítják az osztóhoz viszonyítva, hogy a hányados normál alakban legyen előállítható. Az osztandó módosítása az 5.14. folyamatábrán látható. A törtrész módosításához az szükséges, hogy az osztó törtrésze a TR(TRT) részregiszterben nagyobb legyen, mint az osztandóé az AC(TRT) részregiszterben. A két törtrészt kivonással hasonlítják össze. A kivonás valójában az AC(9Q, TRT) részregiszterben levő törtrész 1-es komplemensével végzett összeadással történik. Az összeadás után, ha az (5.8) által meghatározott Z(Q) bit 1, a TR(TRT) részregiszterben levő törtrész nagyobb, mint az AC(TRT) részregiszterben levő törtrész, és a folyamat tovább folytatódik a karakterisztikák kivonásával és a törtrészek osztásával. Ha a Z(Q) bit 0, akkor a TR(TRT) részregiszterben a törtrész kisebb vagy egyenlő, mint az AC(TRT) részregiszter értéke. Az utóbbi esetben az osztandót egy bitpozícióval jobbra léptetik, hogy a TR(TRT) részregiszterben levő törtrész nagyobbá váljék, mint az AC(9Q, TRT) részregiszterben levő. A jobbra léptetés alatt az AC(9Q) bit helyére egy 1-est iktatnak be, mivel az AC(9Q, TRT) részregiszter az osztandónak vagy a részmaradéknak az 1-es komplemensét tartalmazza. Ugyanakkor az AC(Q, K) részregiszterben a karakterisztikarész eggyel inkrementálódik. Ezután megvizsgálják az AC(Q) bitet lebegőpontos



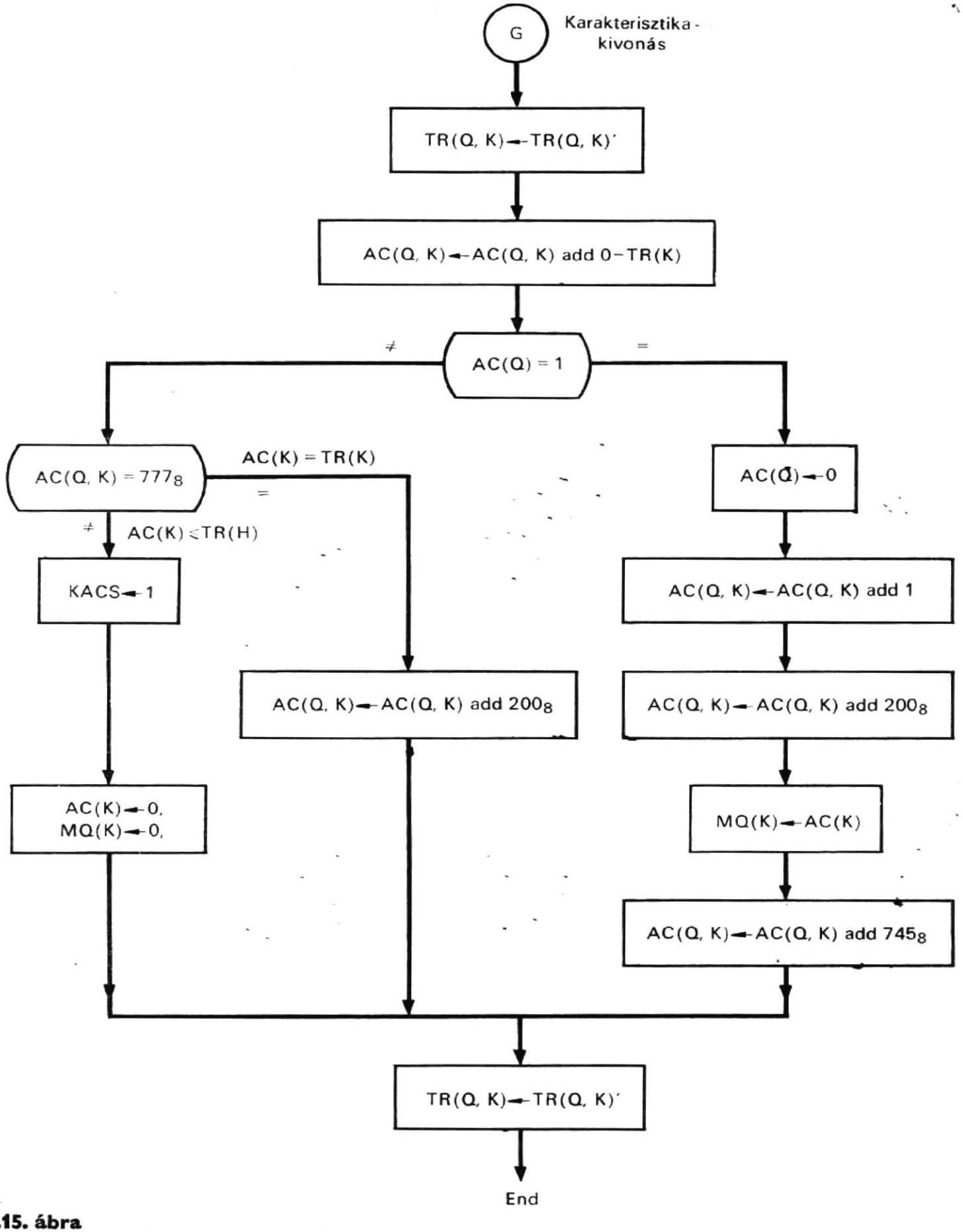
5.14. ábra
Folyamatábra a lebegőpontos osztás osztandó-törtrészének módosításához

túlcsordulásra. Ha túlcsordulás keletkezik, a lebegőpontos osztási folyamat leáll; egyébként pedig folytatódik a karakterisztikák kivonásával, és a törtrészek osztásával.

Az 5.14. ábrán végülis az osztandó törtrésze kisebb, mint az osztóé; és feltételezzük, hogy az osztandó és az osztó kezdetben normál alakban van ábrázolva. Ha az osztandó törtrésze nagyobb, mint az osztóé, a KTCS osztási túlcsordulás-indikátor 1 értéket kap és az osztási folyamat leáll. A törtrész az AC(TRT) részregiszterben változatlan marad.

5.4.3. Karakterisztika-kivonás

A karakterisztika-kivonás a TR(K) részregiszterben levő osztó karakterisztikáját kivonja az AC(Q, K) részregiszterben elhelyezkedő osztandó karakterisztikájából. Az így keletkező hányados-karakterisztika az MQ(K) részregiszterbe kerül. A karakterisztika-kivonás az 5.15. folyamatábrán látható. Minthogy a kivonás a kivonandó 1-es komplemensének az összeadásával hajtódik végre, a TR(K) részregiszterben levő osztó karakterisztikájának először az 1-es komplemensét képezik és azután hozzáadják az



5.15. ábra
Folyamatábra a lebegőpontos osztás karakterisztika kivonási részéhez

AC(Q, K) részregiszterben levő osztandókarakterisztikához. Az eredmény az AC(Q, K) részregiszterbe kerül.

Ezután megvizsgálják az AC(Q) bitet. Ha az AC(Q) bit 1, az osztandó karakterisztikája nagyobb, mint az osztóé és a ciklikus átvitel hozzáadása következik. Ezután a 200_8 eltolást adják hozzá az eredményhez. Most az AC(K) részregiszterben levő karakterisztikát átviszik az MQ(K) részregiszterbe és a maradék karakterisztikájának 27-tel kisebbnek kell lennie, mint az osztandó karakterisztikájának, a 27 kivonását a 33_8 2-es komplemensének (vagyis 745_8) a hozzáadásával végzik el.

Ha az AC(Q) bit 0, az osztandó karakterisztikája egyenlő vagy kisebb, mint az osztó karakterisztikája. Az első esetben az összeg (vagy inkább a különbség) nulla lesz az AC(Q, K) részregiszterben; ehhez azután hozzáadják a 200_8 eltolást. Az utóbbi esetben lebegőpontos alulcsordulás következik be; a KACS alulcsordulás-indikátor 1 értéket kap, az AC(K) és az MQ(K) részregiszter egyaránt 0 lesz.

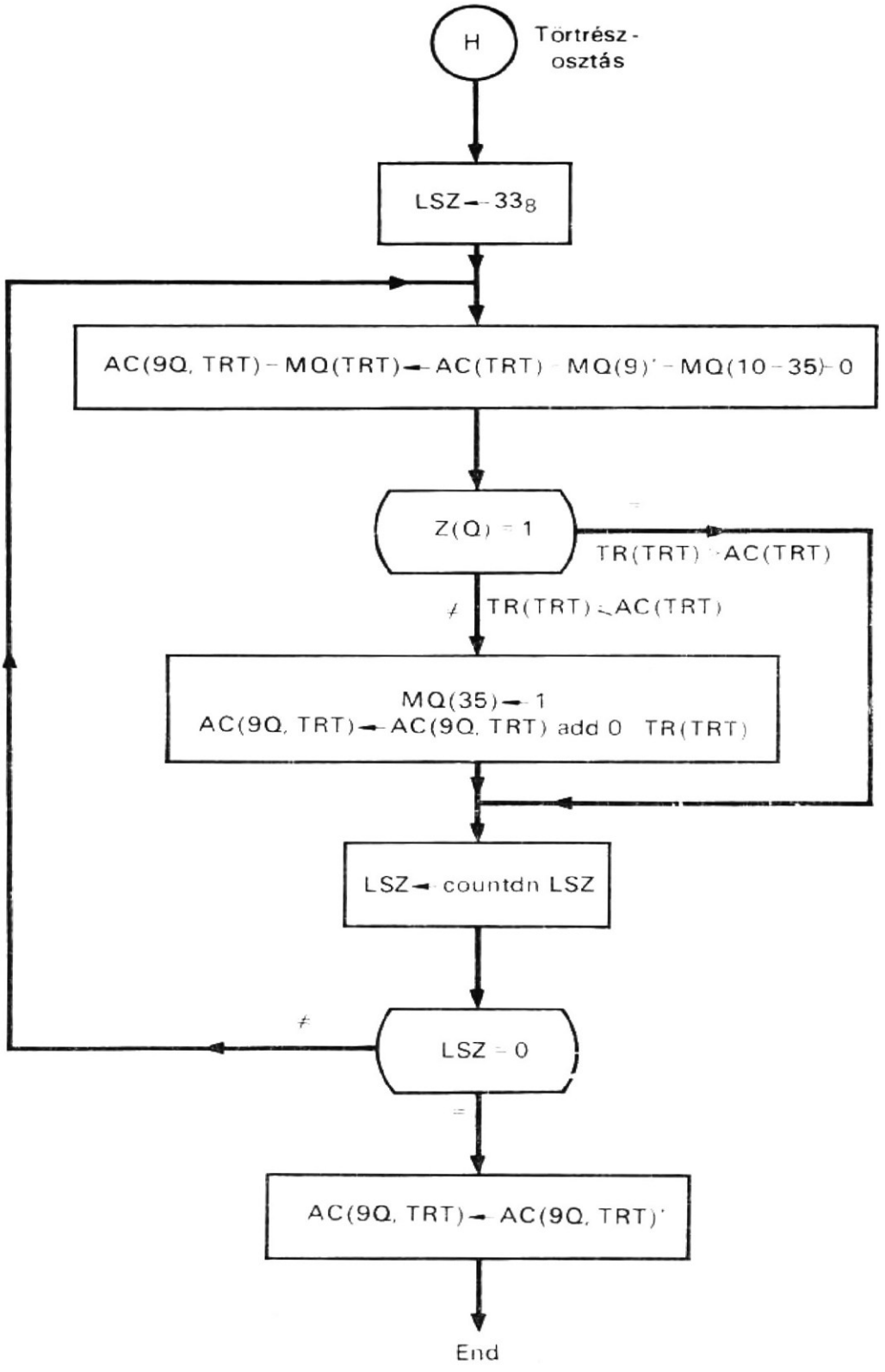
Függetlenül attól, hogy az AC(Q) 1 volt-e vagy 0, az osztó karakterisztikájának ismét 1-es komplemensét képezik, hogy helyes értéke állítódjék elő. Ezzel befejeződik a karakterisztika-kivonás folyamata.

5.4.4. Törtrészek osztása

A törtrészek osztását az 5.16. folyamatábra mutatja. Amint látható, két lebegőpontos szám törtrészének osztása hasonló a korábban leírt fixpontos osztáshoz. Az osztás az LSZ lépésszámlálónak 27-re állításával kezdődik, mivel a törtrész 27-bites. Emlékezzünk vissza, hogy az AC(9Q, TRT) részregiszterben a törtrészt 1-es komplemensét képeztük, az osztandó törtrészeinek módosítása során (l. az 5.14. ábrát). Tehát, a Z(Q) bit jelzi a különbség eredményét. Mivel a TR(TRT) részregiszter törtrésze nagyobb, mint az AC(TRT) részregiszteré, az AC(9Q, TRT)—MQ(TRT) regisztersort 1 bittel balra léptetik. Az MQ(9) bitet komplementálják e léptetés alatt, mivel az AC(9Q, TRT) részregiszterben a törtrész az osztás alatt az 1-es komplemens alakban marad. Ezután összehasonlítják az AC(9Q, TRT) részregiszterben levő törtrészt a TR(TRT) részregiszterben levővel. Ha 0 található az AC(9Q) bit helyén, az AC(TRT)-ben levő törtrész nagyobb vagy egyenlő, mint a TR(TRT)-ben levő; ebben az esetben egy 1-est iktatnak be az M(35) bitre és az AC és TR regiszterek törtrészei összeadódnak. Egyébként az 1-es beiktatása és a törtrészek összeadása nem történik meg. Mindkét esetben eggyel csökkentik az LSZ lépésszámlálót és azután megvizsgálják nulla-e. Ha az LSZ lépésszámláló tartalma nem 0, a fenti lépések: a léptetés balra, vizsgálat, beiktatás, összeadás és lépésszámláló-csökkentés mindaddig folytatódik, amíg az LSZ lépésszámláló értéke nulla nem lesz. Ekkor az AC(9Q, TRT) részregiszterben levő törtrésznek ismét képezik 1-es komplemenseit, és ezzel az osztási folyamat befejeződött.

5.5. CDL leírás

A lebegőpontos összeadás, kivonás, szorzás és osztás algoritmusait eddig folyamatábrákkal írtuk le. Most leírjuk ezeket CDL utasításokkal. Először az időzítő- és vezérlőjeleket írjuk le, majd ezt követi a három műveletsorozat leírása.



5.16. ábra
Folyamatábra a lebegőpontos osztás törtrész-osztási részéhez

5.9.1.
Időzítő- és vezérlőjelek

A lebegőpontos aritmetikai műveletekhez szükséges struktúra csaknem azonos a fixpontos aritmetikai műveletekével. Ezt most az alábbiakban írjuk le:

```
Comment, struktúra, lebegőpontos aritmetikai egység realizálódásához          (5.9)
Register,    D(0—3),                    $időzítőregiszter
             ADDR(0—14),                $a tár címregisztere
             OPKOD(0—3)                 $műveleti kód regisztere
             VSZ(0—3),                  $várakozásszámláló
             I,                         $ha 1, utasításlehívó ciklus;
                                        ha 0, utasításvégrehajtó ciklus
             OLVAS,                     $tár olvasási parancs
             EI,                        $előjelindikátor
             N,                         $vezérlőregiszter
             T,                         $vezérlőregiszter
             W,                         $vezérlőregiszter
             Y,                         $vezérlőregiszter
Memory,      M(ADDR)=M(0—32767, 0—35),
Decoder,     K(0—11)=D,
             J(0—11)=OPKOD,
Switch,      START(ON),
Clock,       P(1—2),
Terminal,    ADKIV=(J(0)+J(1))*I,
             SZOR=J(2)*I,
             OSZT=J(3)*I,
             LAK=(J(4)+J(5))*I,
             LSZ=J(6)*I,
             LOSZ=J(7)*I,
             LA=J(8)*I,
             NO=J(9)*I,
             KK=J(10)*I,
/START(ON)/  D←0,
/P(2)/       D←countup D,
```

A fenti leírásban járulékosan a T regisztert, a VSZ számlálót és új parancsokat deklaráltunk. A LAK, LSZ és LOSZ parancskimenetek (terminálok) sorra, a lebegőpontos összeadás—kivonás, a lebegőpontos szorzás és a lebegőpontos osztáshoz tartozó műveletsorozatot indítják. A LA, NO és KK elnevezésű segédparancsok segédsorozatokat és részsorozatokat vezérelnek.

Az aritmetikai folyamatok leírásában a következőkben valamely utasításvégrehajtó ciklus 12 lépésének idejét ismét a tárciklusidővel egyenlőnek választjuk, és az egyes utasításvégrehajtó ciklusokat úgy választjuk meg, hogy kezdetük egybeessen az egyes tárciklusok kezdetével. Ez a választás a tárat és az aritmetikai egység szinkronizált működését biztosítja az egymást követő ciklusokban. Feltételezzük azt, hogy ha a tár az első lépésben kapott parancsot az olvasásra, a tárból jövő szó a TR regiszterben — a negyedik ciklus végén — fog rendelkezésre állni.

5.5.2.
Lebegőpontos összeadás és kivonás

A lebegőpontos összeadás és kivonás folyamatát az 5.4. és 5.7. folyamatábrákkal írtuk le. A fixpontos műveletekkel összehasonlítva, a lebegőpontos összeadás—kivonás művelet sorozat összetettebbé vált, mivel több további feladatra — így normalizálásra és karakterisztikák azonos értékűvé tételére — van szükség. Bevezetjük a LAK segédparancsot.

A kezdetiérték-adás folyamatábráját az 5.4. ábrán láthatjuk. Ennek a részsorozatnak a 6 lépéséhez a LAK∗K(0)∗P(1), ..., LAK∗K(5)∗P(1) vezérlőjelek szolgálnak. A kezdetiérték-adás CDL leírását az alábbiakban mutatjuk be:

```
Comment, lebegőpontos összeadás és kivonás                    (5.10)
Comment, kezdetiérték-adás                                    (5.11)

/LAK∗K(0)∗P(1)/      OLVAS←1,
/LAK∗K(1)∗P(1)/      N←0, T←0, W←0, Y←0, VSZ←0,
/LAK∗K(2)∗P(1)/      MQ←0,
/LAK∗K(3)∗T(1)/      TR←M(C),
/LAK∗K(4)∗P(1)/      IF (OPKOD=1) THEN (TR(S)←TR(S)'),
/LAK∗K(5)∗P(1)/      IF (TR(S)=AC(S)) THEN (EI←0) ELSE (EI←1),
                     IF (AC(TRT)≠0) THEN
                     (IF TR(TRT)≠0)
                           THEN (N←1, AC(K)←AC(K)',
                           LSZ←27) ELSE (Y←1)),
                     IF (AC(TRT)=0) THEN
                     (IF TR(TRT)=0) THEN (AC←0)
                     ELSE (AC←TR)), Y←1,
/LAK∗K(11)∗P(1)/     IF (Y=1) THEN (Y←0, I←0, D←15)
```

A fenti leírásban, ha nincs nulla törtrész a hatodik lépés alatt, a következő mikroműveletek aktivizálódnak a karakterisztikák azonos értékűvé tételéhez tartozó részsorozat indítására:

```
N←1, AC(K)←AC(K)', LSZ←27,
```

Egyébként az Y regiszter 1 értéket kap és a lebegőpontos összeadási és kivonási sorozat leáll az utasításvégrehajtó ciklus utolsó lépésében.

A karakterisztikák azonos értékűvé tételének folyamatábrája az 5.5. ábrán látható. Ehhez a részsorozathoz a vezérlőjelek a következők: N∗LAK∗K(6)∗P(1), ..., N∗LAK∗K(10)∗P(1). Ahhoz, hogy ebbe a részbe be lehessen lépni, az N-nek 1 értéket kell adni; a rész elhagyásához pedig az N értékének 0-nak kell lennie. Két részsorozat van itt: az egyiket a T regiszter, a másikat pedig a W regiszter vezérli. Ha a két karakterisztika már kiinduláskor azonos, ezt a két részsorozatot megkerülik; ez azzal jár, hogy mind a T, mind a W regisztert 0 értékűnek hagyják. Az AC(9Q, TRT)——MQ(TRT) sorregisztert ismételten léptető húrok a D regiszternek 8-ra állításával kerül végrehajtásra úgy, hogy a következő, P(2) órajel fogja a D regisztert 9-re állítani. A karakterisztikák azonos értékűvé tételének CDL leírása az alábbiakban látható.

```
Comment, karakterisztikák azonos értékűvé tétele              (5.12)

/N∗LAK∗K(6)∗P(1)/    IF (Z(1—8)=377₈)
                     THEN (AC(K)←AC(K)')
                     ELSE (IF (Z(Q)≠1))
                     THEN (AC(K)←AC(K)', T←1) ELSE (W←1),
```

```
/T * N * LAK * K(7) * P(1)/     TR←AC, AC←TR,
/T * N * LAK * K(8) * P(1)/     AC(K)←AC(K)', T←0, W←1,
(W * N * LAK * K(9) * P(1)/     AC(9Q, TRT)—MQ(TRT)←shr
                                AC(9Q, TRT)—MQ(TRT),
                                AC(Q, K)←countdn AC(Q, K),
                                LSZ←countdn LSZ,
(W * N * LAK * K(10) * P(1)/    IF (LSZ=0)+(Z(1—8)=377₈))
                                THEN (AC(K)←AC(K)',
                                W←0, DO VÁRJ)
                                ELSE (D←8),
/N * LAK * K(11) * P(1)/        IF (VSZ=0) THEN (N←0, D←15, OPKOD←8)
                                ELSE (D←10, VSZ←countdn VSZ)
```

A fenti leírásban, ha a karakterisztikák kezdetben már azonos értékűek, a VSZ várakozásszámláló az első lépésben 0 értéket kap, és az N és T regiszterek értéke 0 marad. Ezzel a nyolcadiktól a tizenegyedik lépésig terjedő mikroműveletek elmaradnak és a karakterisztikák azonos értékűvé tételéhez szükséges műveletsorozat leáll az utolsó lépésben.

A karakterisztikák azonos értékűvé tételét be lehet fejezni a végrehajtási ciklus első, harmadik, ötödik, hetedik, kilencedik vagy tizenegyedik lépésében. Kívánatos, és a legtöbb esetben szükséges is, hogy az azonos értékűvé tétel műveletsorozata a végrehajtási ciklus utolsó lépésében álljon le. Erre a célra szolgál a VSZ számláló és az alábbiakban bemutatásra kerülő blokkutasítás.

```
Block, VÁRJ (IF ((LSZ=26)+(LSZ=20)+(LSZ=14)+(LSZ=8)+(LSZ=2))      (5.13)
                                                THEN (VSZ←0),
            IF ((LSZ=25)+(LSZ=19)+LSZ=13)+(LSZ=7)+(LSZ=1))
                                                THEN (VSZ←10),
            IF ((LSZ=24)+(LSZ=18)+(LSZ=12)+(LSZ=6)+(LSZ=0))
                                                THEN (VSZ←8),
            IF((LSZ=23)+(LSZ=17)+(LSZ=11)+(LSZ=5))
                                                THEN (VSZ←6),
            IF((LSZ=22)+(LSZ=16)+(LSZ=10)+(LSZ=4))
                                                THEN (VSZ←4),
            IF ((LSZ=21)+(LSZ=15)+(LSZ=9)+(LSZ=3))
                                                THEN (VSZ←2)),
```

A fenti blokkutasítás egy VÁRJ elnevezésű mikroművelet-blokkot definiál, amely a VSZ számlálónak megfelelő (0, 2, 4, 6, 8 vagy 10) értéket ad. A mikor a részsorozat eléri az utolsó lépést, a VSZ számlálót az (5.12) utolsó utasítása szerint dekrementálják. Ez az utasítás egy-egy utasításból álló hurkot alkot. Annyi lépésszám telik el, hogy amikor a VSZ számláló 0 értékű lesz, a végrehajtási ciklus végére érünk. Ekkor az N regiszter 0 értéket kap és az OPKOD regiszter azt az értéket veszi fel, aminek hatására az LA segédparancsot adja ki.

A törtrész-összeadáshoz és -kivonáshoz, valamint a -normalizáláshoz az 5.6., ill. 5.7. folyamatábra tartozik. A vezérlőjelek: LA * K(0) * P(1),...LA * K(5) * P(1) és LA * K(9) * P(1),..., LA * K(11) * P(1), ahol a parancs az LA segédparancs. Az első három lépés képezi a törtrész-összeadási részsorozatot; a második három lépés a törtrész-kivonási részsorozatot; míg az utolsó három lépés a normalizálási részsorozathoz tartozik. Az alábbiakban látható a törtrész-összeadási és törtrész-kivonási részsorozatok CDL leírása.

```
Comment, törtrész-összeadás                                        (5.14)
/EI' * LA * K(0) * P(1)/   AC(9Q,TRT)←AC(Q,TRT) add 0—TR(TRT),
/EI' * LA * K(1) * P(1)/   IF (AC(9Q)≠1) THEN (Y←1),
```

```
                        ELSE (AC(9Q,TRT)—MQ(TRT)←shr
                        AC(9Q,TRT)—MQ(TRT), AC(Q,K)←countup
                        AC(Q,K), T←1),
/EI′ * LA * K(2) * P(1)/ IF ((AC(Q)=1) * (T=1))
                        THEN (KFTCS←1),*T←0, Y←1,
```

Comment, törtrész-kivonás

```
/EI * LA * K(3) * P(1)/  AC(9Q,TRT)←AC(9Q,TRT)′,
/EI * LA * K(4) * P(1)/  AC(9Q,TRT)←AC(9Q,TRT) add 0—TR(TRT),
/EI * LA * K(5) * P(1)/  IF (AC(9Q)=1) THEN
                         (AC(9Q,TRT)←AC(9Q,TRT) add 1, AC(S)←TR(S), N←1),
                         IF (AC(9Q)≠1) THEN
                         (IF(AC(TRT)=7...7) THEN (AC←0,Y←1)
                         ELSE (AC(9Q,TRT)←AC(9Q,TRT)′, N←1)),
/LA * K(11) * P(1)/      IF (Y=1) THEN (Y←0, I←0, D←15),
```

A fenti leírásban a törtrész-összeadási műveletsorozat akkor kezdődik, ha az EI regiszter 0 értékű; egyébként pedig a kivonási részsorozat kezdődik el. E két részsorozat bármelyike alatt a T regiszter 1 értéket kap, ha a lebegőpontos összeadási vagy kivonási folyamat befejeződött, vagy pedig az N regiszter kap 1 értéket, ha a normalizálási folyamat kezdődik.

A normalizálási részsorozathoz a vezérlőjelek a következők: N * LA * K(9) * P(1),... ...,N * LA * K(11) * P(1). A CDL leírás az alábbiakban látható.

Comment, normalizálás (5.15)

```
/N * LA * K(8) * P(1)/    LSZ←27,
/N * LA * K(9) * P(1)/    IF (AC(9)=1) THEN (MQ←0),
                          IF (MQ(9)=1) THEN
                                       (AC(9Q,TRT)←AC(9Q,TRT) add 1)),
                          IF (AC(9)≠1)  THEN (AC(9Q,TRT)—MQ(TRT)←shl
                                       AC(9Q,TRT)—MQ(TRT),
                                       AC(Q,K)←countdn AC(Q,K),
                                       LSZ←countdn LSZ, W←1),
/W * N * LA * K(10) * P(1)/    IF ((LSZ≠0) * (AC(Q)≠1)) THEN (D←8),
                          IF ((LSZ≠0) * (AC(Q)=1)) THEN (KACS←1, AC←0,
                                       MQ←0,W←0, DO VÁRJ),
                                       IF (LSZ=0) THEN (AC←0, MQ←0, VSZ←8),
/N * LA * K(11) * P(1)/   IF (VSZ=0) THEN (I←0, D←15)
                                       ELSE (D←10, VSZ←countdn VSZ),
                          END
```

A fenti leírásban, ha az összeg normalizált alakban állt elő az összeadás után, nincs szükség normalizálásra. Ebben az esetben a VSZ számláló 0 értéket kap a normalizálási részsorozat első lépésében és a lebegőpontos összeadási és kivonási folyamat az utolsó lépésben leáll. A normalizálás viszont a végrehajtási ciklus első, harmadik, ötödik, hetedik, kilencedik vagy tizenegyedik lépésében fejeződhet be. Abból a célból, hogy a normalizálás pontosan a végrehajtási ciklus utolsó lépésében álljon le, ismét felhasználjuk a VSZ számlálót és a VÁRJ blokkot, a karakterisztikák azonos értékűvé tételénél bemutatott (5.12) leírás szerint.

5.5.3. Lebegőpontos szorzás

A lebegőpontos szorzási folyamatot az 5.9 ... 5.11. folyamatábrákon írtuk le. A lebegőpontos szorzási sorozat vezérlőjelei: LSZR * K(0) * P(1),..., LSZR * K(8) * P(1), a normalizálási részben az N0 segédparancsot használjuk.

A kezdetiérték-adáshoz l. az 5.9. folyamatábrát. E részsorozat hat lépéséhez a vezérlőjelek: LSZR * K(0) * P(1),..., LSZR * K(5) * P(1). A kezdetiérték-adó rész CDL leírását a következőkben láthatjuk:

```
Comment, lebegőpontos szorzás                                        (5.16)

Comment, kezdetiérték-adás                                           (5.17)

/LSZR * K(0) * P(1)/   OLVAS←1,
/LSZR * K(1) * P(1)/   N←0, T←0, W←0, Y←0, VSZ←0,
/LSZR * K(2) * P(1)/   LSZ←27, AC(9Q,TRT)←0, AC(Q,K)←0−MQ(K),
/LSZR * K(3) * P(1)/   TR←M(C),
/LSZR * K(4) * P(1)/   IF (MQ * (S)=TR(S)) THEN (AC(S)←0, MQ(S)←0)
                                           ELSE (AC(S)←1, MQ(S)←1),
/LSZR * K(5) * P(1)/   IF ((TR(TRT)=0)+(MQ(TRT)=0))
                                           THEN (AC←0, MQ←0,Y←1)
                                           ELSE (T←1, N←1),
/LSZR * K(11) * P(1)/  IF (Y=1) THEN (Y←0, I←0, D←15),
```

A fenti leírásban a hatodik lépésben vizsgálják meg, hogy a szorzandó vagy a szorzó törtrésze nem zérus-e? Ha nem, akkor a T regiszter 1 értéket kap, és ez indítja a karakterisztika-összeadási részsorozatot. Továbbá az N regiszter is 1 értéket kap, és ez pedig a törtrész-szorzási részsorozatot indítja. Ha a szorzandó vagy a szorzó törtrésze zérus, az Y regiszter kap 1 értéket és a lebegőpontos szorzási sorozat leáll az aktuális végrehajtási ciklus utolsó lépésében.

A karakterisztika-összeadás folyamatábrája az 5.10.-es. Ehhez a részsorozathoz a vezérlőjelek a következők: T * LSZR * K(6) * P(1),...,T * LSZR * K(8) * P(1). A hhoz, hogy be lehessen lépni a részsorozatba, a T regiszternek 1 értéket kell kapnia; a részsorozat elhagyásához a T regiszter értéke 0 kell, hogy legyen. A karakterisztika-összeadás CDL leírását az alábbiakban mutatjuk be.

```
Comment, karakterisztika-összeadás                                   (5.18)
/T * LSZR * K(6) * P(1)/  AC(Q,K)←AC(Q,K) add 0−TR(K),
/T * LSZR * K(7) * P(1)/  AC(Q,K)←AC(Q,K) add 600₈
/T * LSZR * K(8) * P(1)/  T←0, IF (AC(Q)=1) THEN (KTCS←1, N←0) ELSE (N←1),
/LSZR * K(11) * P(1)/     IF ((N=0) * (T=0)) THEN (I←0, D←15),
```

Ha lebegőpontos túlcsordulás fordul elő, a lebegőpontos szorzási sorozat leáll az aktuális végrehajtási ciklus utolsó lépésében.

A törtrész-szorzási részhez az 5.11. folyamatábra tartozik. Ehhez a részsorozathoz a vezérlőjelek a következők: N * LSZR * K(9) * P(1),..., N * LSZR * K(11) * P(1). Az N regiszter 1 értéket kap a részsorozat indítására, és 0-t kap ezen részsorozat leállítására. A törtrész-szorzás CDL leírása az alábbiakban látható.

```
Comment, törtrész-szorzás                                            (5.19)
/N * LSZR * K(9) * P(1)/  IF (MQ(35)=1) THEN (AC(9Q,TRT)←AC(9Q,TRT)
                                  add 0−TR(TRT)), LSZ←countdn LSZ,
```

```
/N*LSZR*K(10)*P(1)/  AC(9Q,TRT)-MQ(TRT)←shl AC(9Q,TRT)-MQ(TRT),
                     IF (LSZ≠0) THEN (D←8) ELSE (VSZ←5),
/N*LSZR*(11)*P(1)/   IF (VSZ=0) THEN (N←0, OPKOD←9, D←15),
                                 ELSE (D←10, VSZ←countdn VSZ),
```

A fenti összeadást és jobbra léptetést végző hurkot úgy vezérlik, hogy a D regisztternek 8-as értéket adnak. A hurokban 27 interáció van, mivel a törtrész 27-bites. Leírásunk szerint az a végrehajtási ciklus lépés, amelynél a törtrész-szorzás befejeződik, a második lesz. Ezért ebben az ütemben a VSZ számláló 5-ös értéket kap, és így, a VSZ számláló a végrehajtási ciklus nyolcadik lépésében lesz 0 értékű és a rendszer készen áll a normalizálási részsorozatra a következő lépésben.

A normalizálási részhez l. az 5.7. folyamatábrát. A normalizálási részsorozathoz a vezérlőjelek itt a következők: NO*K(9)*P(1),...,NO*K(11)*P(1). A lebegőpontos szorzási folyamat normalizálásához a CDL leírást az alábbiakban láthatjuk.

Comment, normalizálás (5.20)

```
/NO*K(9)*P(1)/          LSZ←27,
                        IF (AC(9)=1)
                        THEN (MQ←0, VSZ←0),
                        IF (MQ(9)=1)
                        THEN (AC(9Q,TRT)←AC(9Q,TRT) add 1),
                        IF (AC(9))≠1)
                        THEN (AC(9Q,TRT)-MQ(TRT)←shl,
                              AC(9Q,TRT)-MQ(TRT),
                              AC(Q,K)←countdn AC(Q,K),
                              LSZ←countdn LSZ, W←1),
/W*NO*K(10)*P(1)/       IF ((LSZ≠0)*(AC(Q)≠1))
                        THEN (D←8),
                        IF ((LSZ≠0)*(AC(Q)=1)),
                        THEN (KACS←1, AC←0, MQ←0, W←0, DO VÁRJ),
                        IF (LSZ=0) THEN (AC←0, MQ←0,W←0, VSZ←8),
/NO*K(11)*P(1)/         IF (VSZ=0) THEN (I←0, D←15)
                        ELSE (D←10, VSZ←countdn VSZ),
                        END
```

A fenti leírás azonos az (5.15)-tel, kivéve, hogy itt a parancs jele NO és az N regisztert nem használjuk.

5.5.4. Lebegőpontos osztás

A lebegőpontos osztási folyamatot az 5.13...5.16. folyamatábrák mutatják. A lebegőpontos osztási sorozat vezérlőjelei a KK és a LOSZ segédparancsot használják fel.

A kezdetiérték-adáshoz l. az 5.13. folyamatábrát. E részsorozat hat lépéséhez a vezérlőjelek a következők: LOSZ*K(0)*P(1),..., LOSZ*K(5)*P(1). A CDL leírása az alábbiakban látható.

Comment, lebegőpontos osztás (5.21)

```
/LOSZ*K(0)*P(1)/    OLVAS←1,
/LOSZ*K(1)*P(1)/    N←0, T←0, W←0, Y←0, VSZ←0,
/LOSZ*K(2)*P(1)/    MQ←0,
```

```
/LOSZ * K(3) * P(1)/      TR←M(C),
/LOSZ * K(4) * P(1)/      IF (TR(S)=AC(S)),
                          THEN (MQ(S)←0),
                          ELSE (MQ(S)←1),
/LOSZ * K(5) * P(1)/      IF (TR(TRT)=0) THEN (OSZTCS←1, Y←1) ELSE
                            (IF (AC(TRT)=0)
                          THEN (MQ←0, AC←0, Y←1)
                          ELSE (N←1)),
/LOSZ * K(11) * P(1)/     IF (Y=1) THEN (Y←0, I←0, D←15),
```

A fenti leírásban a hatodik lépésben vizsgálják meg, hogy az osztó vagy osztandó törtrésze zérus-e, ha nem, akkor az N regiszter 1-es értéket kap, és ezzel elindítja az osztó törtrészének módosítását végző részsorozatot. Ha az osztandó vagy az osztó törtrésze zérus, a V regiszter 1 értéket kap és a lebegőpontos osztási folyamat leáll az aktuális végrehajtási ciklus utolsó lépésében.

Az osztandó törtrészének módosítását az 5.14. folyamatábra mutatja. A vezérlőjelek ehhez a részsorozathoz a következők: N * LOSZ * K(6) * P(1),...,N * LOSZ * K(8) * P(1) és N * LOSZ * K(11) * P(1). Az N regiszter kezdetben 1-es értéket kap, ez indítja ezt a részsorozatot és az N=0 vezérlés pedig leállítja. A CDL leírás az alábbiakban látható:

Comment, osztandó törtrészének módosítása (5.23)

```
/N * LOSZ * K(6) * P(1)/      AC(9Q,TRT)←AC(9Q,TRT)',
/N * LOSZ * K(7) * P(1)/      IF(Z(Q)≠1)
                              THEN (AC(9Q,TRT)—MQ(TRT←
                              1—AC(9Q,9—34)—AC(35)'—M(9—34),
                              AC(Q,K)←countup
                                AC(Q,K), T←1)
                                ELSE (W←1),
/T * N * LOSZ * K(8) * P(1)/ T←0, IF (AC(Q)=1)
                              THEN (KTCS←1, N←0, Y←1,)
                              ELSE (W←1),
/N * LOSZ * K(11) * P(1)/     IF (W=1) THEN (N←0, W←0,
                              OPKOD←10, D←15),
```

A fenti leírásban, ha lebegőpontos túlcsordulás fordul elő, az N regiszter 0, az Y regiszter pedig 1-es értéket kap, és ezzel a lebegőpontos osztási sorozatle áll az aktuális végrehajtási ciklus utolsó lépésében. Egyébként a KK segédparancs generálódik, a karakterisztika-kivonás részsorozat aktiválására.

A karakterisztika-kivonást az 5.15 folyamatábra mutatja. Ehhez a részsorozathoz tartozó vezérlőjelek a következők: KK * K(0) * P(1),..., KK * K(7) * P(1). A CDL leírás az alábbiakban látható:

Comment, a karakterisztika-kivonás (5.24)

```
/KK * K(0) * P(1)/        TR(Q,K)←TR(Q,K)',
/KK * K(1) * P(1)/        AC(Q,K)←AC(Q,K) add 0—TR(K),
/KK * K(2) * P(1)/        IF (AC(Q)=1)
                          THEN (AC(Q)←0, N←1) ELSE (W←1)
                          (IF (AC(Q,K)=777₈)
                          THEN (AC(Q,K)←AC(Q,K) add 200₈)
                          ELSE (KACS←1, AC(K)←0, MQ(K)←0)),
/N * KK * K(3) * P(1)/    AC(Q,K)←AC(Q,K) add 1,
/N * KK * K(4) * P(1)/    AC(Q,K)←AC(Q,K) add 200₈,
/N * KK * K(5) * P(1)/    MQ(K)←AC(K),
```

/N * KK * K(6) * P(1)/ AC(Q,K)←AC(Q,K) add 745_8, W←1, N←0,
(W * KK * K(7) * P(1)/ TR(Q,K)←TR(Q,K)', LSZ←27, W←0,

A fenti leírásban két részsorozat van. Az egyiket az N regiszter vezérli, és akkor aktivizálódik, ha a különbség pozitív, míg a másikat az (5.24)-ben a harmadik végrehajtási utasítás írja le, arra az esetre vonatkozóan, amikor a különbség vagy nulla, vagy negatív. Ez a részsorozat a nyolcadik lépésben áll le és közvetlenül a törtrész-osztás követi az 5.12., 5.15. és 5.16. ábrákon leírt párhuzamos végrehajtás helyett.

A törtrészek osztása az 5.16. folyamatábrán látható. E részsorozat vezérlőjelei a következők: KK * K(8) * P(1),..., KK * K(11) * P(1). A CDL leírást az alábbiakban adjuk meg.

Comment, törtrészek osztása (5.25)

```
/KK * K(8) * P(1)/      AC(9Q,TRT)—MQ(TRT)←AC(TRT)—
                          —MQ(9)'—MQ(10—35)—0,
                        LSZ←countdn LSZ,
/KK * K(9) * P(1)/      IF (Z(Q)=1) THEN (MQ(35)←1,
                          AC(9Q,TRT)←AC(9Q,TRT)
                          add 0—TR(TRT)),
/KK * K(10) * P(1)/     IF (LSZ≠0) THEN (D←7)
                          ELSE (AC(9Q,TRT)←AC(9Q,TRT)', VSZ←7),
/KK * K(11) * P(1)/     IF (VSZ=0) THEN (I←0, D←15)
                          ELSE (D←10, VSZ←countdn VSZ),
                        END
```

A fenti leírásban, az AC(9Q,TRT)—MQ(TRT) regisztersort ismételten balra léptető, az osztót összeadó és az LSZ lépésszámlálót dekrementáló hurkot a D regiszternek 7-es értékre állításával kapjuk és a legközelebbi, P(2) órajel ad a D regiszternek 8-as értéket. Miután 27-szer végighaladt a hurkon, a törtrész-osztás az aktuális végrehajtási ciklus negyedik lépésében befejeződik. Annak érdekében, hogy ez a részsorozat pontosan a végrehajtási ciklus utolsó lépésében álljon le, a VSZ számláló 7-es értéket kap. A VSZ számlálót ezután az egyutasításos hurokban addig csökkentik, amíg értéke nulla nem lesz. Ekkor befejeződött a lebegőpontos osztási folyamat.

Feladatok

5.1. Írjunk proceduális CDL leírást, az 5.4...5.7. folyamatábrákkal meghatározott lebegőpontos összeadásra és kivonásra!

5.2. Ismételjük meg az 5.1. feladatot az 5.9...5.11. folyamatábrákkal leírt lebegőpontos szorzásra!

5.3. Ismételjük meg az 5.1. feladatot, az 5.13...5.16. folyamatábrákkal leírt lebegőpontos osztásra!

5.4. Fortran vagy más algoritmikus nyelv felhasználásával írjunk programot az 5.4... 5.7. folyamatábrákkal leírt lebegőpontos összeadás és kivonás szimulálására és ellenőrzésére!

5.5. Ismételjük meg az 5.4. feladatot, az 5.9... 5.11. folyamatábrákon leírt lebegőpontos szorzás szimulálására és ellenőrzésére!

5.6. Ismételjük meg az 5.4. feladatot az 5.13. és 5.16. folyamatábrákkal meghatározott lebegőpontos osztásra!

5.7. Módosítsuk a (4.37) terminálutasítással leírt csoport- és szekcióátvitelű párhuzamos összeadót úgy, hogy lebegőpontos aritmetikai egységhez fel lehessen használni!

5.8. Fortran vagy más algoritmikus nyelv falhasználásával szimuláljuk és ellenőrizzük az 5.7. feladatban leírt párhuzamos összeadót!

5.9. Mi azoknak a végrehajtási ciklusoknak a maximális és minimális száma, amelyek az (5.10)...(5.15) utasításokban leírt lebegőpontos összeadáshoz és kivonáshoz szükségesek?

5.10. Mi azoknak a végrehajtási ciklusoknak a maximális és minimális száma, amelyek az (5.16)...(5.20) utasításokban leírt lebegőpontos szorzáshoz szükségesek?

5.11. Mi azoknak a végrehajtási utasításoknak a maximális és minimális száma, amelyek az (5.21)...(5.25) utasításokban leírt lebegőpontos osztáshoz szükségesek?

5.12. Válasszuk meg a mikroprogramtárat, a mikroutasítás formátumát és a többi szükséges funkcionális elemet, majd készítsünk mikroprogramot:

a) az (5.10)...(5.15) utasításokkal leírt lebegőpontos összeadási és kivonási művelet sorozathoz;

b) az (5.10)...(5.20) utasításokkal leírt lebegőpontos szorzási műveletsorozathoz;

c) az (5.21)...(5.25) utasításokban leírt lebegőpontos osztási műveletsorozathoz úgy, hogy ezeket a műveletsorozatokat a vezérlőtárban levő mikroprogram hajtsa végre!

5.13. A VÁRJ blokkot az (5.13) utasítás definiálta. Használjuk a következő mikroműveletet:

VSZ←2 count12 VSZ,

az (5.12) és (5.15) utasításokban szereplő VÁRJ blokk helyettesítésére! A count12 speciális operátor, amely moduló 12-ben számlál.

Irodalom

[1] BURKS, A.W.—GOLDSTINE, H. H.—von NEUMANN, J.: *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument.* 1946, Reprinted in Datamation 8, No. 9, pp. 24...31, September, 1962, and 8, No. 10, pp. 36...41, October, 1962.

[2] PHISTER, M., JR.: *Logical Design of Digital Computers.* New York, John Wiley & Sons, Inc., 1958.

[3] METROPOLIS, N.—ASHENHURST, R. L.: *Significant Digit Computer Arithmetic.* IRE Transactions of Electronic Computers, December, 1958, pp. 265...267.

[4] BLAAUW, G. A.: *Indexing and Control-Word Technique.* IBM Journal of Research and Development 3, July, 1959, pp. 288...301.

[5] GARY, H. L.—HARRISON, C, JR.: *Normalized Floating-point Arithemetic with an Index of Significance.* Proc. of the Eastern Joint Computer Conference, 1959, pp. 244...248.

[6] ASHENHURST, R. L.—METROPOLIS, N.: *Unnormalized Floating-point Arithmetic.* J. of the ACM, July, 1959, pp. 415...428.

[7] CARR, JOHN W. III: *Error Analysis in Floating-point Arithmetic.* Comm. of the ACM, May, 1959, pp. 10...15.

[8] WADLY, W. G.: *Floating-point Arithmetics.* J. of the ACM, April, 1960, pp. 129...139.

[9] BECHMANN, F. S.—BROOKS, F. P., JR.—LAWLESS W. J., JR.: *Developments in the Logical Organization of Computer Arithmetic and Control Units.* Proc. of the IRE 49, No. 1, pp. 53...66, January, 1961.

[10] MacSORLEY, O. L.: *High-Speed Arithmetic in Binary Computers,* Proc. of the IRE 49, No. 1, ppl. 67...91, January, 1961.

[11] BUCHHOLZ, W.: *Planning a Computer Systems.* New York, McGraw-Hill Book Company, 1962.

[12] CHU, Y.: *Digital Computer Design Fundamentals.* New York, McGraw-Hill Book Company, 1962.

[13] FLORES, I.: *The Logic of Computer Arithmetic.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1963.

[14] STERBENZ, P. H.: *Floating-point Number Systems.* Notes, June, 1966.

[15] STERBENZ, P. H.: *UBM 7094 Principles of Operation.* Form A22-6703-4, 5th Edition, October, 21, 1966.

[16] FLORES, I.: *Computer Design,* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1967.

[17] GSCHWIND, H., W.: *Design of Digital Computers.* Springer-Verlag, New York, Inc., 1967.

[18] KNUTH, D. E.: *The Art of Computer Programming Fundamental Algorithms.* vol. 2. Reading, Massachusetts, Addison-Wesley Publishing Co., Inc., 1969.

[19] CHU, Y.: *Introduction to Computer Organization.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1970.

6.
Soros aritmetikai egységek

Az aritmetikai egységek általában összeadás, kivonás, szorzás és osztás végrehajtására alkalmasak. Ezeket a műveleteket párhuzamosan, sorosan vagy ezeket kombinálva lehet elvégezni. A párhuzamos aritmetikai egység két szám számjegyeit egyidejűleg adja össze, a soros aritmetikai egység viszont két szám összeadását számjegyről-számjegyre végzi el, egy egyszerűbb felépítésű összeadóval. Tehát a párhuzamos aritmetikai egység rövidebb időt igényel az összeadási műveletek végrehajtására, viszont a soros aritmetikai egység realizálása egyszerűbb. Ez a fejezet soros aritmetikai egységekkel foglalkozik, az előző 4. és 5. fejezetek párhuzamos aritmetikai egységeket írnak le.

Ebben a fejezetben egy bináris, soros aritmetikai egységet és egy decimális, számjegyként soros aritmetikai egységet mutatunk be. A bináris, soros aritmetikai egység egyszerre egy biten, vagy egyszerre több biten, más elnevezéssel bittöbbesen (például két biten) hajt végre aritmetikai műveletet (pl. összeadást). Tehát egybites vagy több bites összeadó—kivonóra lesz szükségünk. Hasonlóképpen a decimális aritmetikai egység egyetlen decimális számjegyen vagy decimális számjegyek többesén tud egyszerre pl. összeadást végrehajtani. Ehhez egydecimális számjegy összeadóra vagy több decimális számjegy összeadóra lesz szükségünk. Az ebben a fejezetben leírt bináris aritmetikai egység egybites összeadót vagy kivonót, a decimális aritmetikai egység pedig egydecimális számjegy összead—kivonót használ föl.

6.1.
Bináris, soros aritmetikai egység struktúrája

A bináris, soros aritmetikai egység struktúrájának meghatározása a bináris számformátumának megválasztásával kezdődik. Ezután az összeadó/kivonót specifikáljuk, amit a többi szükséges funkcionális elem kiválasztása követ. A vezérlőrész elemeinek kiválasztását viszont későbbre halasztjuk és az időzítő- és vezérlőjelekkel együtt tárgyaljuk.

6.1.1.
Fixpontos számábrázolás

Az aritmetikai egységhez 24-bites bináris számot választunk, előjeles 2-es komplemens ábrázolásban [4]. A formátum a 6.1. ábrán látható, ahol a bináris pont az előjelbit és a legértékesebb számbit között helyezkedik el; tehát a szám törtszám és fixpontos.

Ha a szám pozitív, akkor előjellel és abszolút értékkel ábrázolható, vagyis:

$$X=0\times 2^0+\sum_{i=1}^{23} x_1\cdot 2^{-i}, \tag{6.1}$$

ahol X a szám, 0 a pozitív előjelet ábrázolja, az x_i-k pedig a számbitek. Az ábrázolható legnagyobb szám 0,11...1 vagy $(2^0...2^{-23})$. Ha a szám negatív, a negatív számot 2-es komplemens kódban ábrázoljuk, vagyis felírhatjuk:

$$X=1\times 2^1-\{0\times 2^0+\sum_{i=1}^{23} x_i\cdot 2^{-1}\}, \tag{6.2}$$

ahol 1 reprezentálja a negatív előjelet. Minthogy a szám 2-es komplemense egyenlő a szám 1-es komplemensének és a legkisebb helyiértékű számbitnek az összegével, a fenti ábrázolást a következőképpen is felírhatjuk:

$$X=1\times 2^0+\sum_{i=1}^{n} x_i'\cdot 2^{-i}+2^{-n} \tag{6.3}$$

és

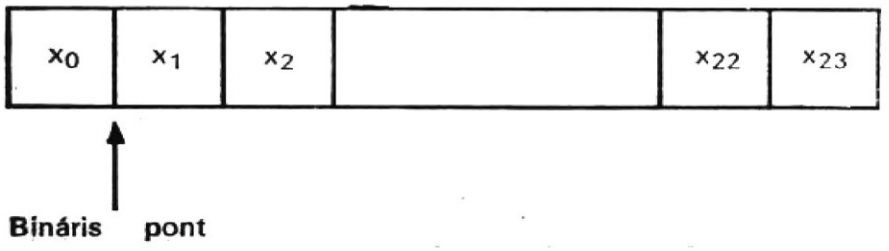
$$x_i'=1-x_i,$$

ahol x_i' az x_i jegy 1-es komplemense. A legkisebb így ábrázolható negatív szám 1,000...0, vagy -1.

6.1.2.
Teljes összeadó—kivonó

Az egybites teljes összeadó olyan logikai hálózat, amelynek három bemenete és két kimenete van. Legyenek a bemenetek X, Y és W_i, amelyek sorra az összeadót, az összeadandót és a bemeneti átvitelt képviselik. Legyenek a kimenetek Z és W_0. Ezek sorrendben az összeget és a kimeneti átvitelt jelentik. Az egybites teljes összeadót a következő terminál- (bemenő kapocspár-) utasítással lehet leírni.

$$\begin{aligned} \text{Terminal,}\quad & Z = X\oplus Y\oplus W_i, \\ & W_0 = X*Y+Y*W_i+W_i*X. \end{aligned} \tag{6.4}$$



6.1. ábra
A bináris szám formátuma

Az egybites teljes kivonó olyan logikai hálózat, amelynek szintén három bemenete és két kimenete lehet. Legyen a bemenetek és kimenetek jelölése X, Y, W_i; Z és W_0. Az egybites teljes kivonót a következő terminál- (bemenő kapocspár-) utasítással lehet leírni:

Terminal, $Z = X \oplus Y \oplus W_i$ (6.5)
$W_0 = X' * Y + Y * W_i * X'$.

Amint a fentiekben látható, a Z kimenet logikai egyenlete ugyanaz mindkét terminálutasításban, azonban a W_i átvitelt jelöl a (6.4) utasításban és átvételt a (6.5) utasításban. A W_0 kimenet logikai egyenlete is hasonló, az eltérés az, hogy az X bemenet, amely a kivonandót jelöli, 1-es komplemensével (tagadott értékével) szerepel a (6.5)-ben.

A fenti összeadót és kivonót egyetlen logikai hálózatban össze lehet vonni. Az N egybites vezérlőregiszter összeadási parancsot ad, ha 1-est tartalmaz és kivonási parancsot, ha a tartalma 0. Az egybites teljes összeadó—kivonót a következő terminálutasítással lehet leírni:

Terminal, $Z = X \oplus Y \oplus W_i$ (6.6)
$W_0 = (N \odot X) * Y + (N \odot X) * W_i + Y * W_i$,

Ha az N regiszter 1-est tartalmaz, a (6.6) terminálutasításból a (6.4) utasítás lesz; ha az N regiszter tartalma 0, a (6.5) utasítással egyenértékű. A (6.6) utasítással leírt összeadó—kivonó egyike azoknak a funkcionális elemeknek, amelyeket a bináris, soros aritmetikai egységben használni fogunk.

6.1.3. Struktúra

A bináris, soros aritmetikai egységhez kiválasztott funkcionális elemeket a vezérlési rész kivételével, a 6.2. ábra blokkdiagramja mutatja, ahol a TAK jelöli az egybites teljes összeadó—kivonót. Amint látható, az A regiszter az akkumulátor, Q regiszter a szorzó—hányados regiszter és az R regiszter egy másik operandusregiszter, amely egyúttal a tár tartalomregisztere is. Az előbbiekben leírt összeadó—kivonóval együtt ez a három regiszter hajtja végre az aritmetikai műveleteket. A 6.1. táblázat ban láthatjuk ennek a három regiszternek a funkcióját összeadás, kivonás, szorzás és osztás közben; ezeket a funkciókat később tárgyaljuk.

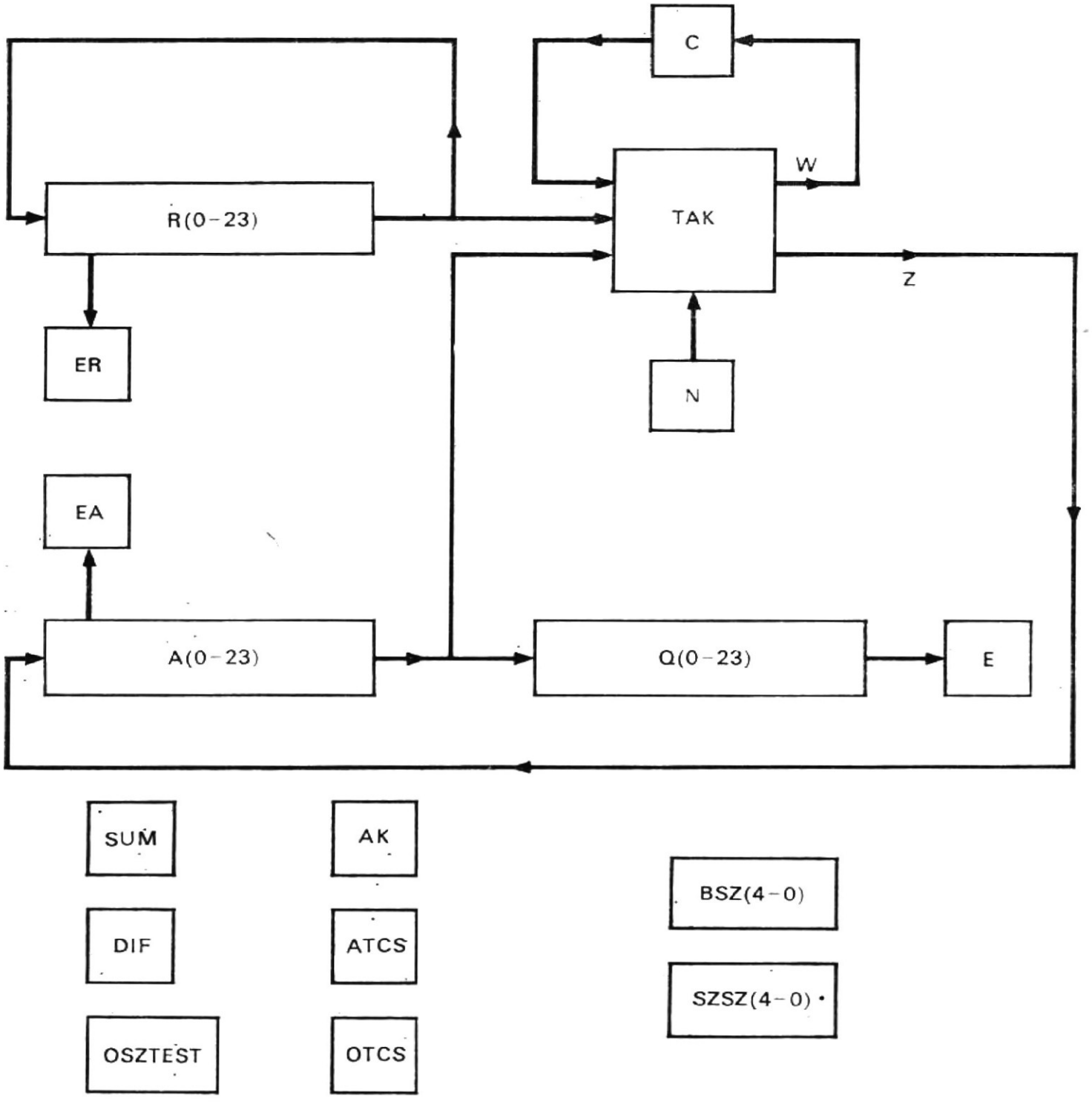
Két 4-bites számláló van: a BSZ és az SZSZ, továbbá tizenegy egybites regiszter: ER, EA, AK, ATCS, OTCS, SUM, DIF, N, C, E és OSZTEST. Az A regiszter A(0), ill. az R regiszter R(0) előjelbitjét az EA, ill. az ER regiszterek tárolják. Az AK regiszter jelzi az összeadás vagy a kivonás előtti speciális átvitelt vagy átvételt. Az ATCS regiszter összeadás alatti túlcsordulást, míg az OTCS regiszter osztás alatti túlcsordulást jelez. A SUM, DIF és OSZTEST regisztereket arra használjuk, hogy az összeadási részsorozatot, a kivonási részsorozatot, ill. az osztás-stop ellenőrzési részsorozatot kezdeményezzék. Az N regiszter hajtja végre az összeadás—kivonás vezérlést. A C regiszter tárolja az átvitelt vagy az átvételt. Az E regisztert átmeneti tárolásra használják a szorzáskor és osztáskor.

A fenti struktúra a következő deklarációs utasításokkal írható le:

Comment, bináris, soros aritmetikai egység-struktúrája (6.7)

Register, R(0—23), $tartalomregiszter
A(0—23), $akkumulátor

Q(0—23),	$szorzó—hányados regiszter
BSZ(4—0),	$bitszámláló
SZSZ(4—0),	$szószámláló
E,	$átmeneti regiszter
C,	$átvitelregiszter
ATCS,	$összeadási túlcsordulás-indikátor
OTCS,	$osztási túlcsordulás-indikátor
N,	$összeadás—kivonás vezérlőregiszter (összeadás történik, ha értéke 1)
SUM,	$vezérlőregiszter az összeadási részsorozathoz
DIF,	$vezérlőregiszter a kivonási részsorozathoz



6.2. ábra
A bináris, soros aritmetikai egység struktúrája (nem tartalmazza a vezérlőrészt)

6.1. táblázat

A regiszterek funkciói az aritmetikai műveletek alatt

Művelet	R regiszter	A regiszter		Q regiszter	
		induláskor	befejezéskor	induláskor	befejezéskor
összeadás	összeadandó	összeadandó	összeg	nem használt	nem használt
kivonás	kivonandó	kisebbítendő	különbség	nem használt	nem használt
szorzás	szorzandó	nulla	szorzat*	szorzó	szorzat**
osztás	osztó	osztandó	maradék	nulla	hányados

* A szorzat nagyobb helyiértékű része.
** A szorzat kisebb helyiértékű része.

```
                OSZTEST,    $vezérlőregiszter az osztás-stop ellenőrzési részsorozathoz
                AK,         $átmeneti tároló az átvitel vagy átvétel tárolására
                EA,         $az A(0) előjel tárolása
                ER,         $az R(0) előjel tárolása
Casregister,    AQE(0-48)=A-Q-E,
                AQ(0-47)=A-Q,

Comment, az egybites összeadó-kivonó leírása

Terminal,       Z=R(23)⊕A(23)⊕C,                $összegterminál
                W=(N⊙A(23))*R(23)+
                +(N⊙A(23))*C+
                +R(23)*C,                       $átvitel- vagy átvételterminal

Comment, összeadás vagy kivonás alatti túlcsordulás-ellenőrző hálózat kimenete

Terminal,       ATCSTEST = N*EA'*ER'*C+
                         +N*EA*ER*C'+
                         +N'*EA'*ER*C'+
                         +N'*EA*ER'*C,

Comment, végérték osztás alatti osztás-stop ellenőrzéshez

Terminal,       OSZTSTOP = N'*A(0)'*R(0)'*EA*E'+
                         +N'*A(0)*R(0)*EA'+
                         +N*A(0)'*R(0)*EA*E'+
                         +N*A(0)*R(0)'*EA'.
```

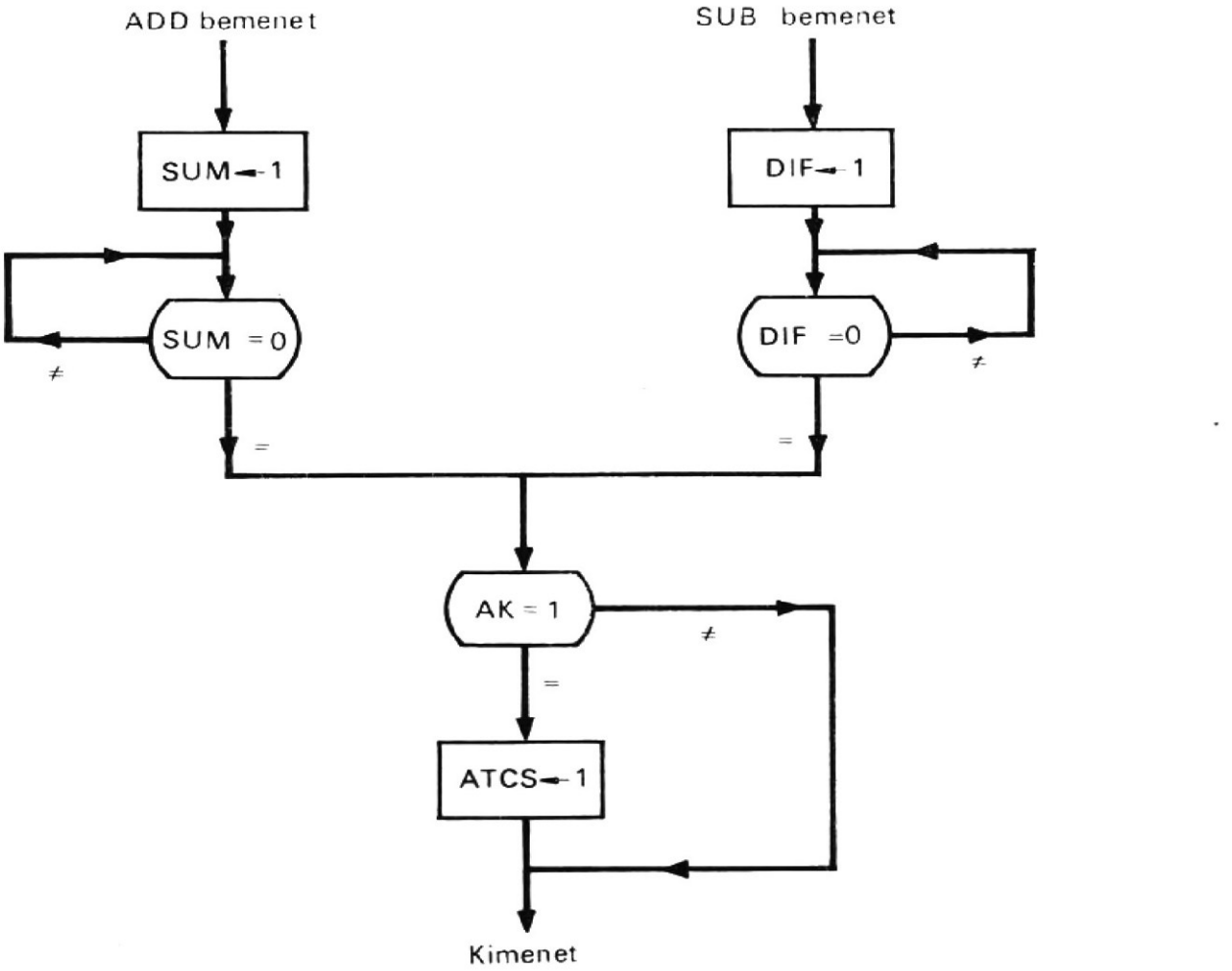
A fenti leírásban az AQE regisztersort a szorzás, az AQ regisztersort pedig az osztás elvégzéséhez deklaráltuk. Az első terminálutasítás az egybites teljes összeadó—kivonót definiálja, ahol a bemenetek A(23), R(23) és C, a kimenetek pedig Z és W. A második terminálutasítás az összeadás és kivonás alatti túlcsordulás ellenőrzéséhez tartozó hálózat kimenetét írja le. A harmadik terminálutasítás az osztás alatti osztás-stop feltétel ellenőrzésére használt hálózat kimenetét írja le. Az egyenleteket a későbbiekben igazoljuk.

6.2. Bináris összeadás és kivonás

A soros aritmetikai egységben a bináris összeadáshoz és kivonáshoz a direkt összeadási és kivonási algoritmust alkalmazzuk. Az alábbiakban bemutatjuk az algoritmust, a túlcsordulási feltételeket, a struktúrát és a folyamatábrát.

6.2.1. Algoritmus

Az előjeles, 2-es komplemens ábrázolású bináris számok összeadására és kivonására használható algoritmust [4] a 6.3. folyamatábrán mutatjuk be. Amint látható, ez összeadáskor összeadja a két összeadandót, és kivonáskor kivonja a kivonandót a kisebbítendőből. Mind összeadáskor, mind kivonáskor az előjelbitet számbitként kezelik. Ha a két szám előjele megegyezik, pl. az összeadás esetén, vagy különböző, pl. kivonáskor, túlcsordulás következhet be; ezt ezután ellenőrzik. Ezzel az összeadás vagy a kivonás befejeződött.



6.3. ábra
Folyamatábra előjeles 2-es komplemens ábrázolású bináris számok összeadásához és kivonásához

6.2.2.
Túlcsordulási feltételek

Ha két, 23-bites, azonos előjelű bináris számot összeadnak, vagy két, különböző előjelű számot kivonnak egymásból, akkor az eredmény abszolút értéke nagyobb lehet, mint az ábrázolható legnagyobb szám. Ha ez az állapot előfordul, akkor *túlcsordulás* keletkezik. Ha túlcsordulás van, az összeg vagy a különbség használhatatlanná válik.

Az előjeles, 2-es komplemens ábrázolású bináris számok összeadásakor vagy kivonásakor előforduló túlcsordulási feltételeket a 6.2. táblázat mutatja. Amint látható, két bináris szám összeadásakor akkor fordul elő túlcsordulás, ha átvitel jelenik meg

6.2. táblázat

Túlcsordulási feltételek összeadáskor és kivonáskor (előjeles, 2-es komplemens ábrázolású bináris számoknál)

Eset	Összeadás vagy kivonás	Előjelbitek		Túlcsordulási feltétel*
		A(0)	R(0)	
A	összeadás	0	0	átvitelt generált az lhb
B	összeadás	1	1	nem generált átvitelt az lhb
C	kivonás	0	1	nem generált átvitelt az lhb
D	kivonás	1	0	átvitelt generált az lhb

* lhb=legnagyobb helyiértékű bit.

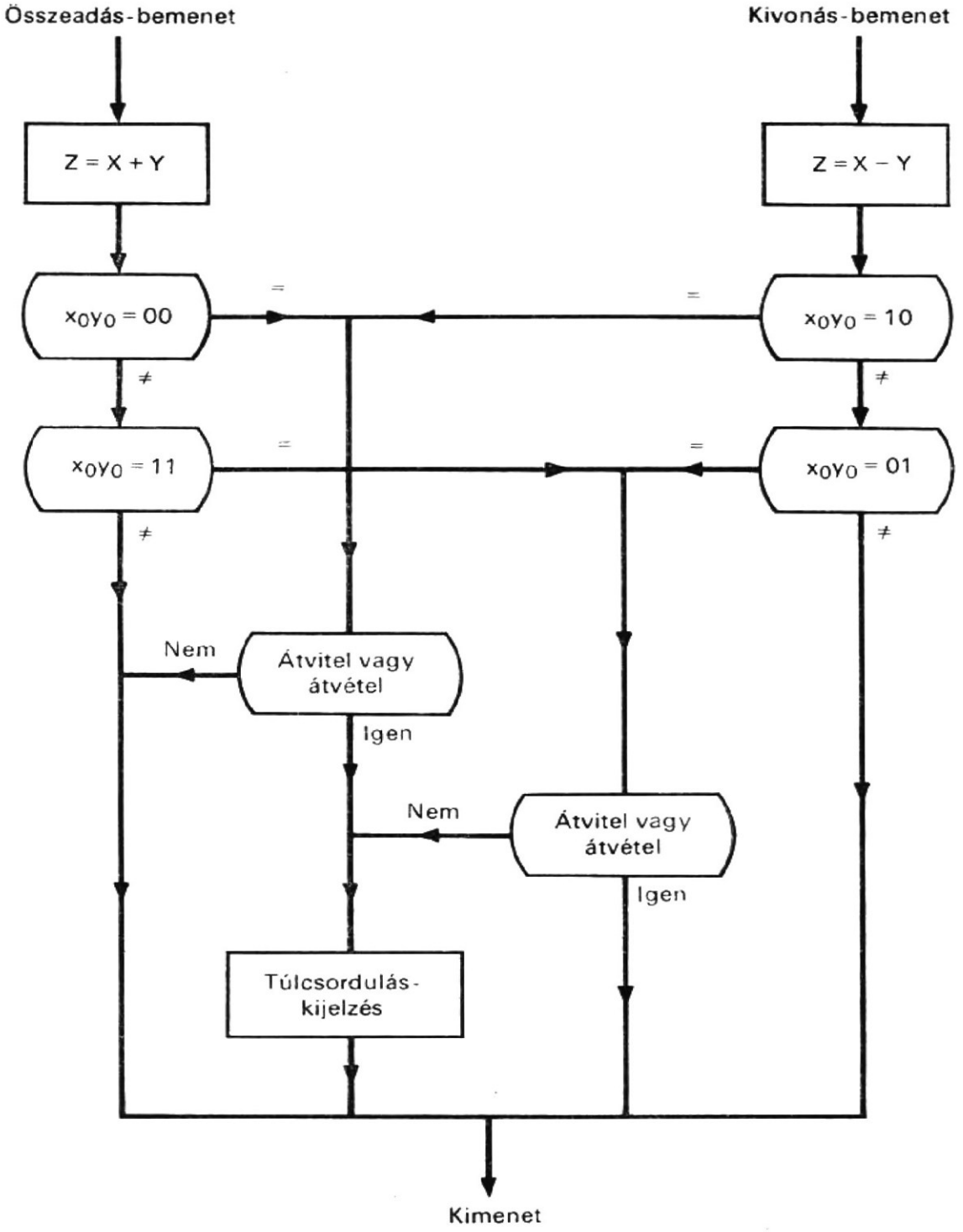
a legnagyobb helyiértékű számbitről, ha mindkét szám pozitív; vagy ha nem jelenik meg átvitel a legnagyobb helyiértékű számbitről, ha mindkét szám negatív. Amikor két bináris számot kivonunk egymásból, túlcsordulás jön létre, ha átvétel jelenik meg a legnagyobb helyiértékű (legértékesebb) számbit felől, ha a kisebbitendő negatív és a kivonandó pozitív; vagy ha nem jelenik meg átvétel a legnagyobb helyiértékű számbit felől, ha a kisebbítendő pozitív és a kivonandó negatív.

Ha a fenti túlcsordulási feltételeket beiktatjuk a 6.3. folyamatábrába, akkor eredményül a 6.4. folyamatábrát kapjuk. Ezen az ábrán a Z az összeg vagy a különbség, X az összeadandó vagy a kisebbítendő, Y az összeadandó vagy a kivonandó és x_0 és y_0 az X, ill. az Y előjelét reprezentálja. Az átvitel vagy átvétel a legértékesebb számbit felől értelmezendő.

A soros aritmetikai egységben a két szám előjelét az EA és ER regiszter tárolja. Az átvitelt vagy átvételt a C regiszter tartalmazza. Az összeadást vagy kivonást az N regiszter vezérli. Tehát a fentiekben meghatározott túlcsordulási feltételek abban a terminálutasításban jelennek meg, amely az ATCSTEST kimenetet definiálja a (6.7) utasításban, és ez használható az összeadás és kivonás alatti túlcsordulás ellenőrzésére.

6.2.3.
Struktúra

Az összeadáshoz és a kivonáshoz az A és az R regisztert használjuk. Amint a 6.1. táblázatból látható, az egyik összeadandót vagy a kisebbítendőt kezdetben az A regiszter tárolja, a másik összeadandót és a kivonandót pedig az R regiszter. Összeadás, ill. kivonás után az összeg, ill. a különbség az A regiszterbe kerül, az összeadandó és a kisebbítendő pedig elvész.



6.4. ábra
Folyamatábra előjeles 2-es komplemens ábrázolású bináris számok összeadásához és kivonásához

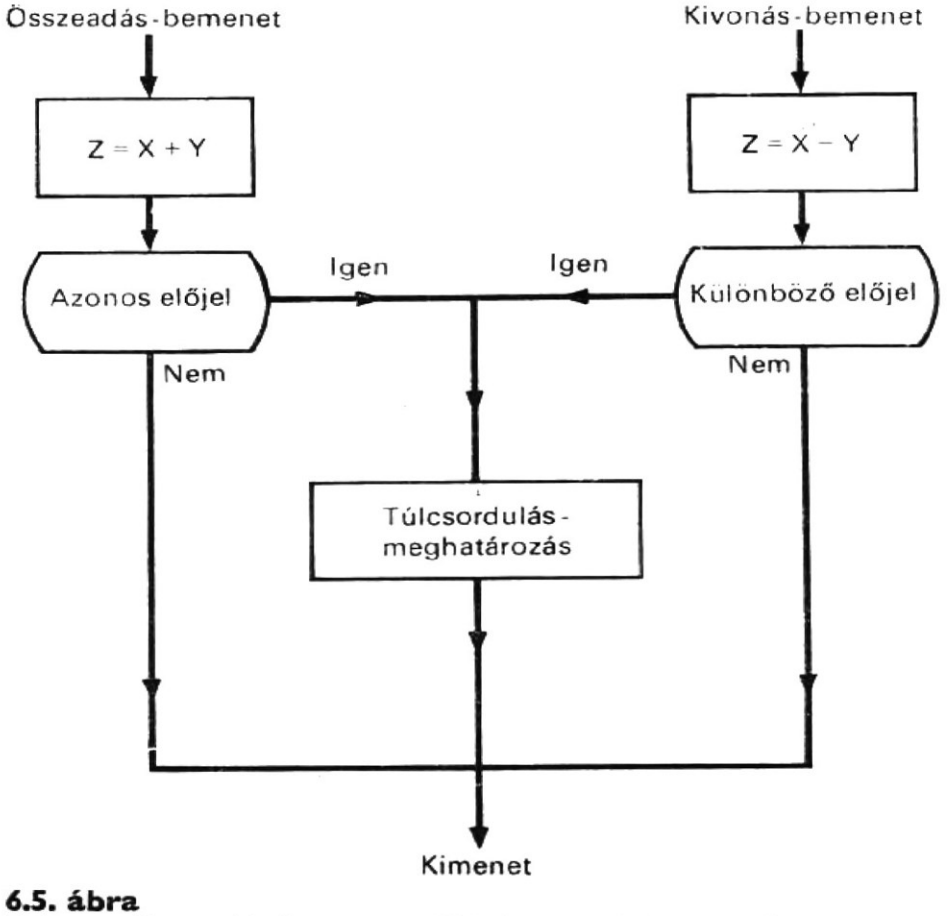
Az AK regiszter jelzi, hogy van-e átvitel vagy átvétel a legértékesebb bitből, a 6.2. táblázat feltételei szerint. Ez jelzi a túlcsordulást összeadás és kivonás esetén. Ha túlcsordulás fordul elő, az ATCS regiszter 1 értéket kap a túlcsordulás kijelzésére. Ennek az átvitelnek vagy átvételnek a jelenléte vagy hiánya viszont nem jelez túlcsordulást szorzás vagy osztás esetén, bár összeadás és kivonás a szorzás és osztás alkalmával is felhasználásra kerül. Az AV regiszternek ez esetben nem kell 1 értéket kapnia, mivel túlcsordulás nem így jön létre osztáskor.

6.2.4. Folyamatábra

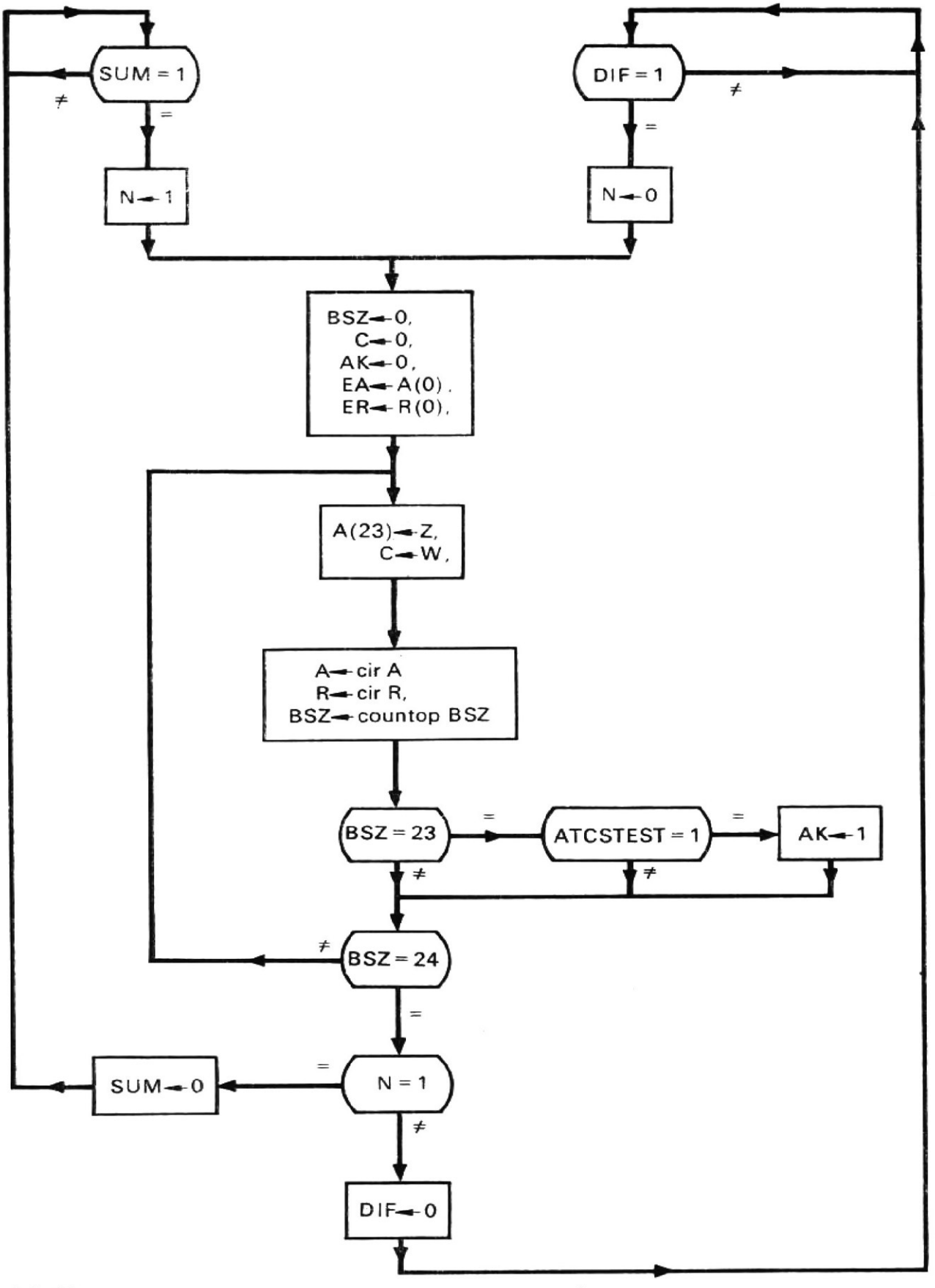
Az összeadási és kivonási sorrend a 6.5. folyamatábrán látható. A 6.6. folyamatábra pedig a SUM-DIF részsorozatot mutatja, ezt „hívják be" az összeadás és kivonás elvégzése folyamán, amikor a SUM vagy a DIF regiszter 1 értéket kap (6.5. ábra).

Amint a 6.6. ábrán látható, a SUM-DIF kezdetű részsorozat vizsgálja meg a SUM és a DIF regisztereket egy-egy várakozási hurokban. Amikor a SUM vagy a DIF regiszter 1 értéket kap, az ábrának megfelelően 1, ill. 0 érték kerül az N regiszterbe. Mindkét esetben azután a részsorozat kezdetiérték-adása történik meg oly módon, hogy a BSZ, C és AK regiszterek 0 értéket kapnak és az A(0) bitet az EA, az R(0) bitet pedig az ER regiszter tárolja. Ezután belépünk a bitösszeadási hurokba. Az egybites összeadó—kivonóról jövő Z és W kimenetek először beíródnak az A(23), ill. a C bitekbe. Ezután az A és az R regisztert egy bitpozícióval ciklikusan jobbra léptetik, majd a BSZ számláló értéke 1-gyel megnő. Amikor a BSZ számláló eléri a 23-at, megvizsgálják az ATCSTEST kimenetet, van-e túlcsordulás. Ha van túlcsordulás, az AK regiszter 1 értéket kap. Ezt követően megvizsgálják a BSZ számlálót 24 értékre. Ha tartalma nem 24, a részsorozat visszatér az összeadó—kivonóból jövő kimenetek tárolásához. Ez a bitösszeadási hurok addig ismétlődik, amíg a BSZ számláló el nem éri a 24 értéket. Ekkor a SUM vagy a DIF regiszter 0 értéket kap, attól függően, hogy az N regiszter tartalma 1, ill. 0. Most már befejeződött a SUM-DIF részsorozat végrehajtása, és a részsorozat visszatér a várakozási hurokhoz.

A 6.6. ábrán az összeadási részsorozat azzal kezdődik, hogy a SUM regiszter 1 értéket kap, a kivonási sorozat pedig azzal, hogy a DIF regiszter kap 1 értéket és ezzel indítják a SUM-DIF részsorozatot; ezután mindegyik várakozik a befejezésére. A rész-



6.5. ábra
Folyamatábra a bináris összeadási és kivonási sorozathoz



6.6. ábra
Folyamatábra a SUM-DIF részsorozathoz

sorozat akkor fejeződik be, amikor a SUM vagy a DIF regiszter tartalma 0. Az összeadási és kivonási sorozat ezután azzal folytatódik, hogy megvizsgálja, vajon tartalmaz-e az AK regiszter 1-et. Ha 1-et tartalmaz, az ATCS regiszter 1-et kap, hogy jelezze az összeadási túlcsordulást. Ezzel befejeződött az összeadás vagy a kivonás.

6.3. Bináris szorzás

A bináris szorzás a soros aritmetikai egységben a Booth-féle algoritmust használja fel. Az alábbiakban bemutatjuk az algoritmust, a struktúrát és a folyamatábrát.

6.3.1. Algoritmus

A Booth-féle algoritmus [4] a 6.7. folyamatábrán látható, ahol X a szorzandó, P a részlet etszorzat, i az index és n 23-mal egyenlő. Az Y a következő alakú:

$$Y = y_0 \times 2^0 + \sum_{i=1}^{n} y_i 2^{-i}. \tag{6.8}$$

Amint a 6.7. ábrából látható, hogy ez az algoritmus két szomszédos: az y_i és y_{i+1} bitet vizsgálja egyidejűleg. Ha ez a két bit 10, az új részletszorzat úgy jön létre, hogy kivonjuk az X szorzandót az aktuális részletszorzatból, vagyis:

$$P \leftarrow P - X$$

Ha a bitpár 01, akkor az új P részletszorzat oly módon keletkezik, hogy az X szorzandót hozzáadjuk az aktuális részletszorzathoz, azaz:

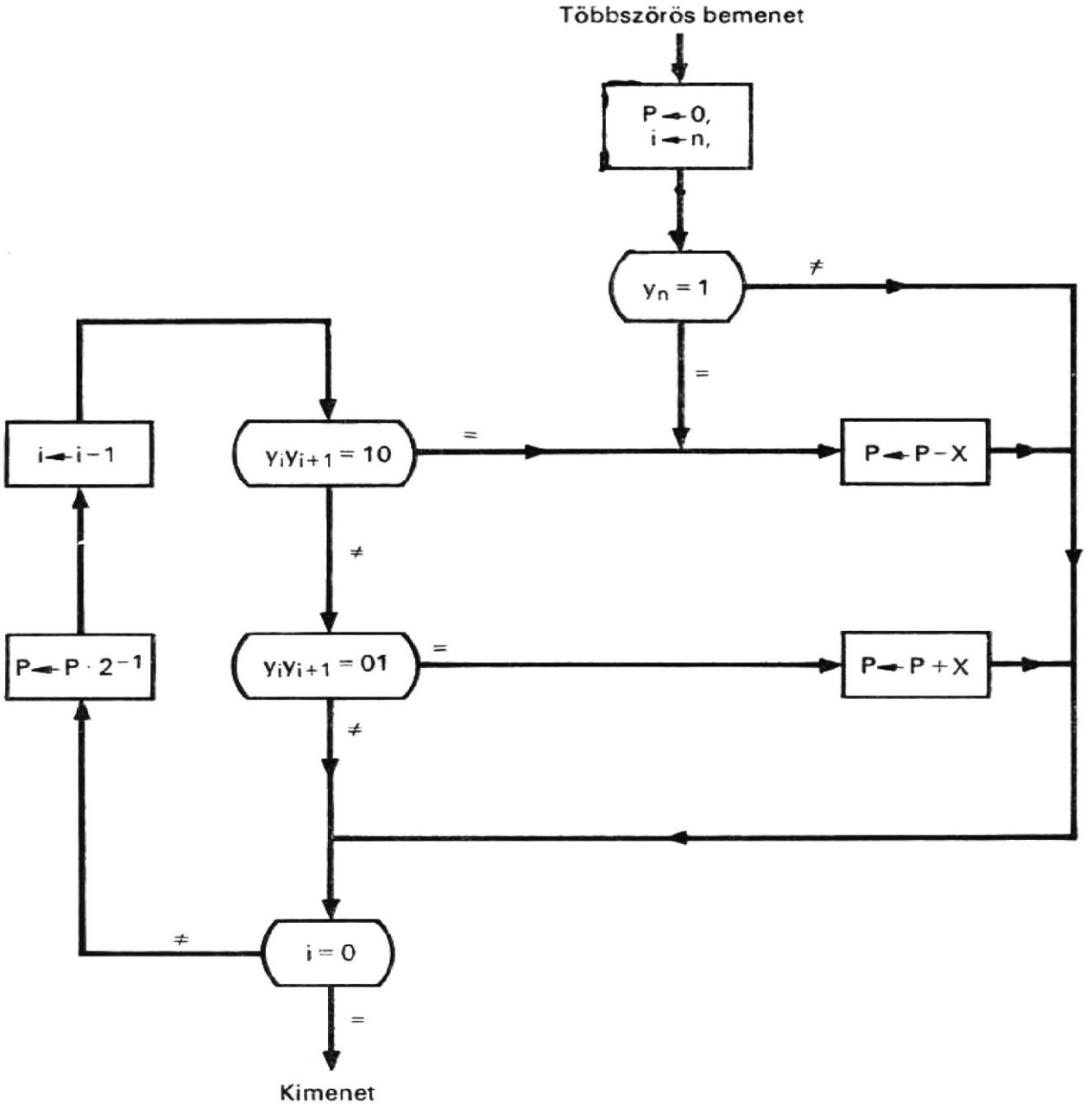
$$P \leftarrow P + X$$

Ha a bitpárok 00 vagy 11, a P részletszorzatot sem nem adják össze, sem nem vonják ki. A három eset mindegyikében azután 2-vel osztják a P részletszorzatot és az i indexet 1-gyel csökkentik. Két másik szomszédos bitet vizsgálnak meg hasonlóan és egy további részletszorzatot képeznek. A folyamat addig folytatódik, amíg az i el nem éri a 0-t. Ekkor a szorzás befejeződött.

Amikor a szorzás elkezdődik, csak egy bitet, az y_n bitet vizsgálják meg. Ha az 1, az első P részletszorzatot úgy képezik, hogy az X szorzandót kivonják a kezdeti részletszorzatból, amint az a 6.7. ábrán látható; egyébként pedig sem összeadás, sem kivonás nem történik. Amint a 6.7. ábrából is látható, az utolsó részletszorzat (vagyis a szorzat) létrehozása után nem történik 2-vel osztás.

6.3.2. Struktúra

A Boothe-algoritmussal végzett bináris szorzás az A, R, Q és E regisztereket használja. Amint a 6.1. táblázatból látható, a szorzandót és a szorzót kezdetben az R, ill. a Q regiszter tárolja, és az A és az E regiszter kiindulásakor 0. Szorzás közben az A, Q és E regiszterek regisztersort képeznek a jobbra léptetéshez. A szorzás után a szorzatot az A és a Q regiszter tárolja oly módon, hogy a szorzat nagyobb helyiértékű része az A, a kisebb helyiértékű része pedig a Q regiszterben foglal helyet.



6.7. ábra
A Booth-algoritmussal végzett szorzást bemutató folyamatábra, előjeles 2-es komplemens ábrázolású bináris számokhoz

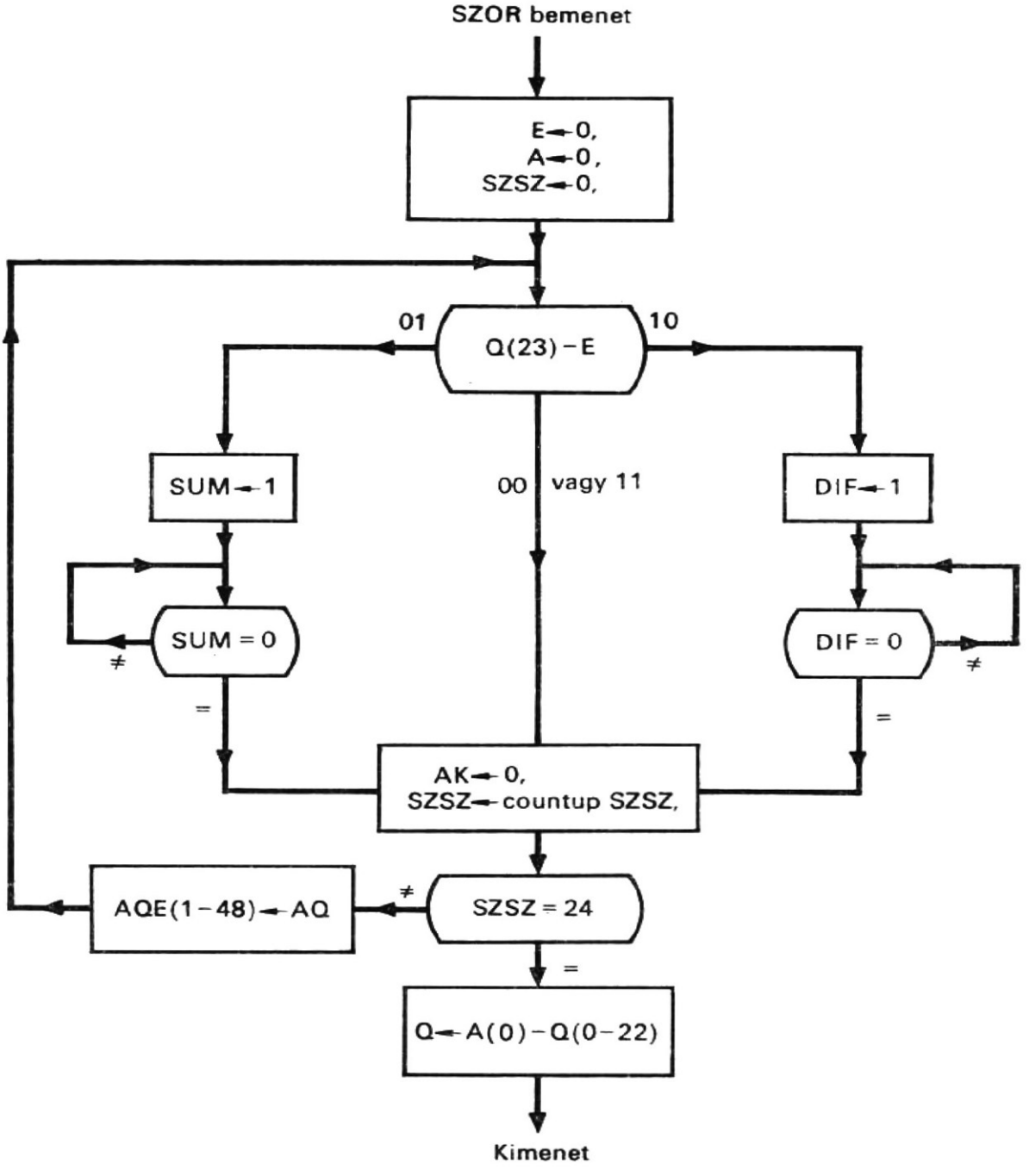
6.3.3. Folyamatábra

A szorzás műveletsorozatot a 6.8. folyamatábrán láthatjuk. Minthogy a szorzásban összeadásra és kivonásra egyaránt szükség van, a szorzási műveletsorozatban a 6.6. ábrán bemutatott SUM-DIF részsorozatot hívják be az összeadás és kivonás végrehajtására.

Amint a 6.8. ábrából látható, az E, az A és az SZSZ regiszter kezdetben 0 értéket kap. Az ismételt összeadási hurok a Q(23) és az E bitek egyidejű vizsgálatával kezdődik. Ha ezek értéke 01, ill. 10, összeadást, ill. kivonást kell végezni. (Az első alkalommal csak 00 vagy 10 fordulhat elő, mivel az E regiszter kezdetben 0 értéket kapott.) Mindkét esetben ezután az AK regiszter 0 értéket kap és az SZSZ számlálót 1-gyel megnövelik. Ezt követően megvizsgálják az SZSZ számlálót, hogy a tartalma már

24-e. Ha nem, az AQE regisztersort egy bitpozícióval jobbra léptetik és a folyamat ismét visszatér a hurok elejére, majd a Q(23) és az E bitek ismételt vizsgálatával folytatódik. Ez a folyamat addig ismétlődik, amíg az SZSZ számláló el nem éri a 24 értéket. Ekkor végezetül léptetés jobbra mikroművelet kerül még végrehajtásra, azért, hogy a szorzat kisebb helyiértékű részét a Q(1—23) részregiszter tárolhassa. A szorzás ezzel befejeződött.

A 6.8. ábrán, amikor a SUM részsorozatot hívták be az összeadás, vagy a DIF részsorozatot a kivonás végrehajtására, a szorzási sorozat megvárja a részsorozat befejezését, miközben állandóan megvizsgálja a SUM vagy a DIF regisztert. Megjegyezzük, hogy e részsorozat végrehajtása és a SUM vagy a DIF regiszter vizsgálata szimultán megy végbe. Ilyen szimultán műveletvégzés történik akkor is, amikor a SUM-DIF részsorozatot hívják be a szorzási és az osztási művelet sorozatba.



6.8. ábra
Folyamatábra a bináris szorzási művelet sorozathoz

6.4. Bináris osztás

A bináris osztás elvégzésére a Burks, Goldstine és von Neumann [1] által kidolgozott, visszaállítás nélküli (nonrestoring) algoritmust használjuk fel. Az alábbiakban bemutatjuk az algoritmust, a struktúrát, az osztás-stop feltételt és a folyamatábrákat.

6.4.1. Algoritmus

A visszaállítás nélküli (nonrestoring) algoritmust részleteiben már másutt [4] leírtuk. Röviden, az algoritmus részmaradékot képez oly módon, hogy az osztót minden alkalommal összeadja vagy kivonja, valahányszor hányadosbit (vagy még inkább: pszeudohányados-bit) generálódik. Pontosabban legyen X és Y sorrendben az osztandó és az osztó. A részmaradék a következő egyenlet szerint jön létre:

$$r_i = 2r_{i-1} + (1 - 2q_i)Y. \tag{6.9}$$

Ehhez járul még az a szabály, hogy ha az r_{i-1} (nem r_i) maradék előjele és az Y előjele megegyezik, a q_i hányadosbit 1, és az r_i részmaradékot úgy kapjuk meg, hogy az Y osztót kivonjuk — (6.9) szerint — a $2r_{i-1}$-ből. Ha ez a két előjel különböző, a q_i hányadosbit 0, és az r_i részmaradékot úgy nyerjük, hogy az Y osztót hozzáadjuk a $2r_{i-1}$-hez. A Q hányados a q_i hányadosbiteknek az alábbi módon végzett összegezéséből adódik:

$$Q = (-1 + 2^{-n}) + \sum_{i=1}^{n} q_i 2^{-(i-1)}, \tag{6.10}$$

ahol $(-1+2^{-n})$ a hányadosbitekhez hozzáadott korrekció. (Ez a magyarázata, hogy a q_i-ket miért nevezzük pszeudohányados-biteknek.)

A visszaállítás nélküli (nonrestoring) algoritmust a 6.9. folyamatábra mutatja. Amint látható, a kiindulási maradék az X osztandó. Az ábrán levő hurokban megvizsgálják az r_{i-1} részmaradék és az Y osztó előjelét, ezek alapján generálják a q_i hányadosbitet, összegezik a Q hányadost, új maradékot képeznek, megnövelik az i indexet és azután ellenőrzik azt a hurokból való kilépés vizsgálatára. A kilépés után hozzáadják a korrekciós tagot a Q hányadoshoz, így megkapják a végső hányadost.

6.4.2. Osztás-stop állapot

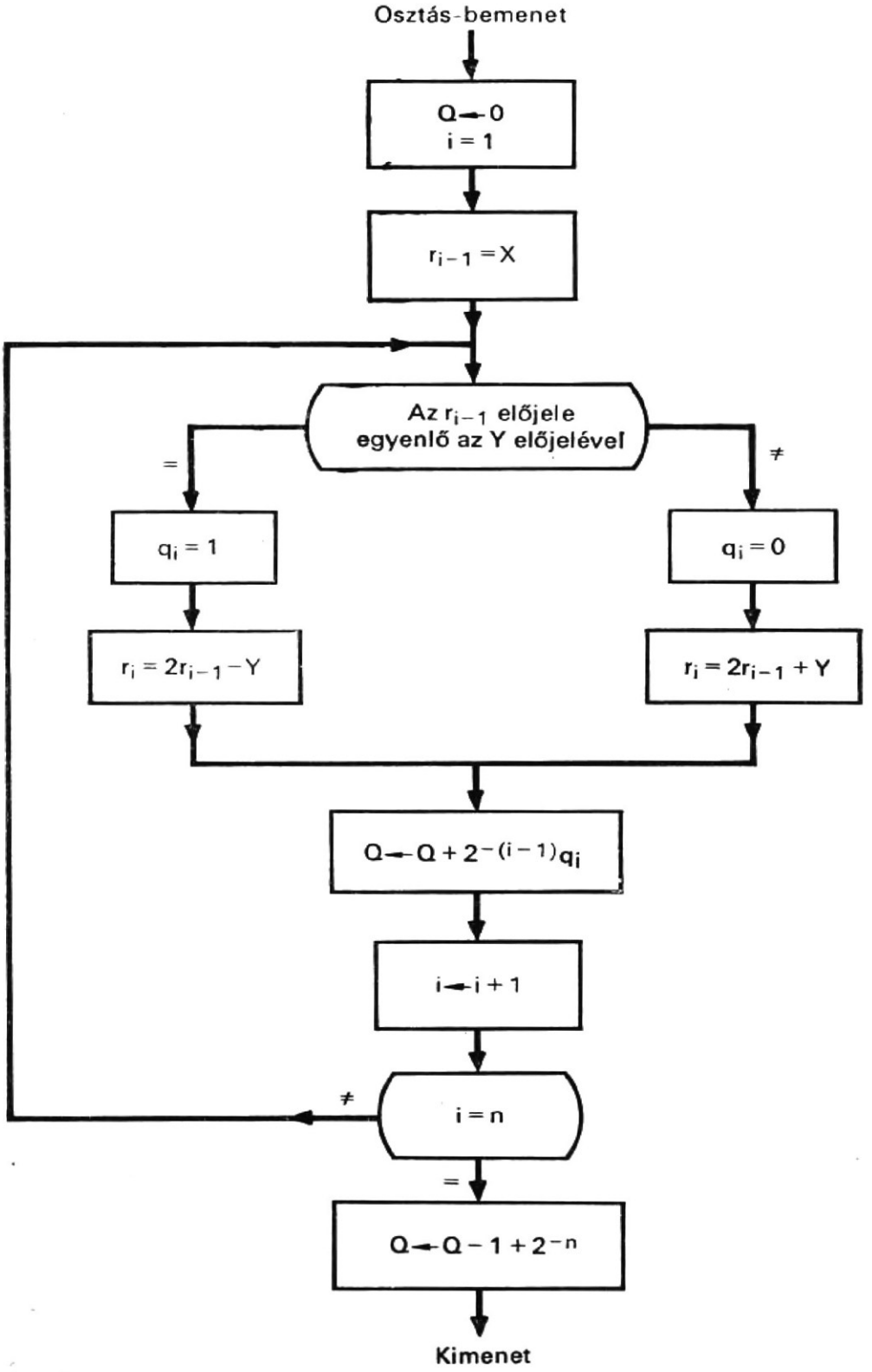
Ha az osztó az osztandóhoz viszonyítva túl kicsi, a hányados túlságosan nagy lesz ahhoz, hogy a Q regiszterben tárolni lehessen. Amikor ilyen helyzet fordul elő, osztási túlcsordulás következik be; a hányados pontatlan és az osztást le kell állítani.

Amint az előzőkben bemutattuk, az osztandót és az osztót törtszámoknak választottuk. Rendkívül kívánatos, hogy a hányados is törtszám legyen. Ez a választás ahhoz a kritériumhoz vezet, hogy az osztandónak kisebbnek kell lennie, mint az osztó. Ebből meghatározható az osztás-stop állapot. Négy eset határozza meg az osztás-stop állapotot, ezek a 6.3. táblázatban láthatók. Az esetek az osztandó és az osztó elő-

jeleitől függnek. Ha a két előjel azonos, kivonást kell végrehajtani (ez az *a*) és a *b*) eset) annak ellenőrzésére, vajon az osztandó kisebb-e, mint az osztó; egyébként összeadás szükséges az ellenőrzéshez (ez a *c*) és a *d*) eset). Kivonás esetén az osztót kell kivonni az osztandóból. A négy eset a következő:

a) Ha mindkét előjel pozitív, osztási túlcsordulás keletkezik, ha a különbség pozitív vagy nulla. Ez abból állapítható meg, hogy a különbség előjele pozitív.

b) Ha mindkét előjel negatív (a számok most előjeles, 2-es komplemens ábrázolásban vannak), osztási túlcsordulás keletkezik, ha a különbség negatív vagy 0. Ez abból állapítható meg, hogy a különbség előjele negatív, vagy a különbség nagysága 0.



6.9. ábra
A visszaállítás nélküli (nonrestoring) algoritmussal végzett osztást bemutató folyamatábra, előjeles 2-es komplemens ábrázolású bináris számokhoz

c) Ha az osztandó előjele pozitív, de az osztóé negatív, osztási túlcsordulás jön létre, ha az összeg pozitív vagy nulla. Ebben az esetben az összeg előjele pozitív.

d) Ha az osztandó előjele negatív, de az osztóé pozitív, osztási túlcsordulás keletkezik, ha az összeg negatív vagy nulla. Ez abból állapítható meg, hogy az összeg előjele negatív vagy az összeg nagysága 0.

A (6.7) harmadik terminál- (kapocspár-) utasítása az OSZTSTOP végértéket definiálja, amely ezt a négy, a 6.3. táblázatban található osztás-stop feltételt tartalmazza.

6.3. táblázat

Osztás—stop ellenőrzési feltételek (számábrázolás: előjeles, 2-es komplemens)

Eset	Előjelbit		Művelet	Osztási túlcsordulás van, ha	Ellenőrzési feltétel*
	A(0)	R(0)			
a	0	0	kivonás	a különbség pozitív vagy 0	az előjel +
b	1	1	kivonás	a különbség negatív vagy 0	az előjel − vagy az érték 0
c	0	1	összeadás	az összeg pozitív vagy 0	az előjel +
d	1	0	összeadás	az összeg negatív vagy 0	az előjel − vagy az érték 0

* Az előjel és az érték az összegre vagy a különbségre vonatkozik.

Ezt a terminálutasítást a következőképpen kaptuk. A szükséges összeadást vagy kivonást az N, ill. N′ reprezentálja, mivel az N regiszter összeadást, ill. kivonást vezérli. Az osztandó és az osztó előjeleit sorra az A(0), ill. az R(0) bit jelöli. Azt, hogy az összeg vagy a különbség előjele pozitív- vagy negatív-e, az EA′, ill. EA határozza meg, mivel az EA regiszter tárolja az összeg vagy a különbség előjelét. Azt, hogy az összeg vagy a különbség értéke nulla-e vagy sem, az E regiszter mutatja, amint látni fogjuk. Tehát a 6.3. táblázatban szereplő *a*) és *c*) esetre az osztás-stop állapotokat a következő Boole-függvényekkel lehet reprezentálni:

$$N' * A(0) * R(0) * EA' \quad \text{és} \quad N * A(0) * R(0)' * EA$$

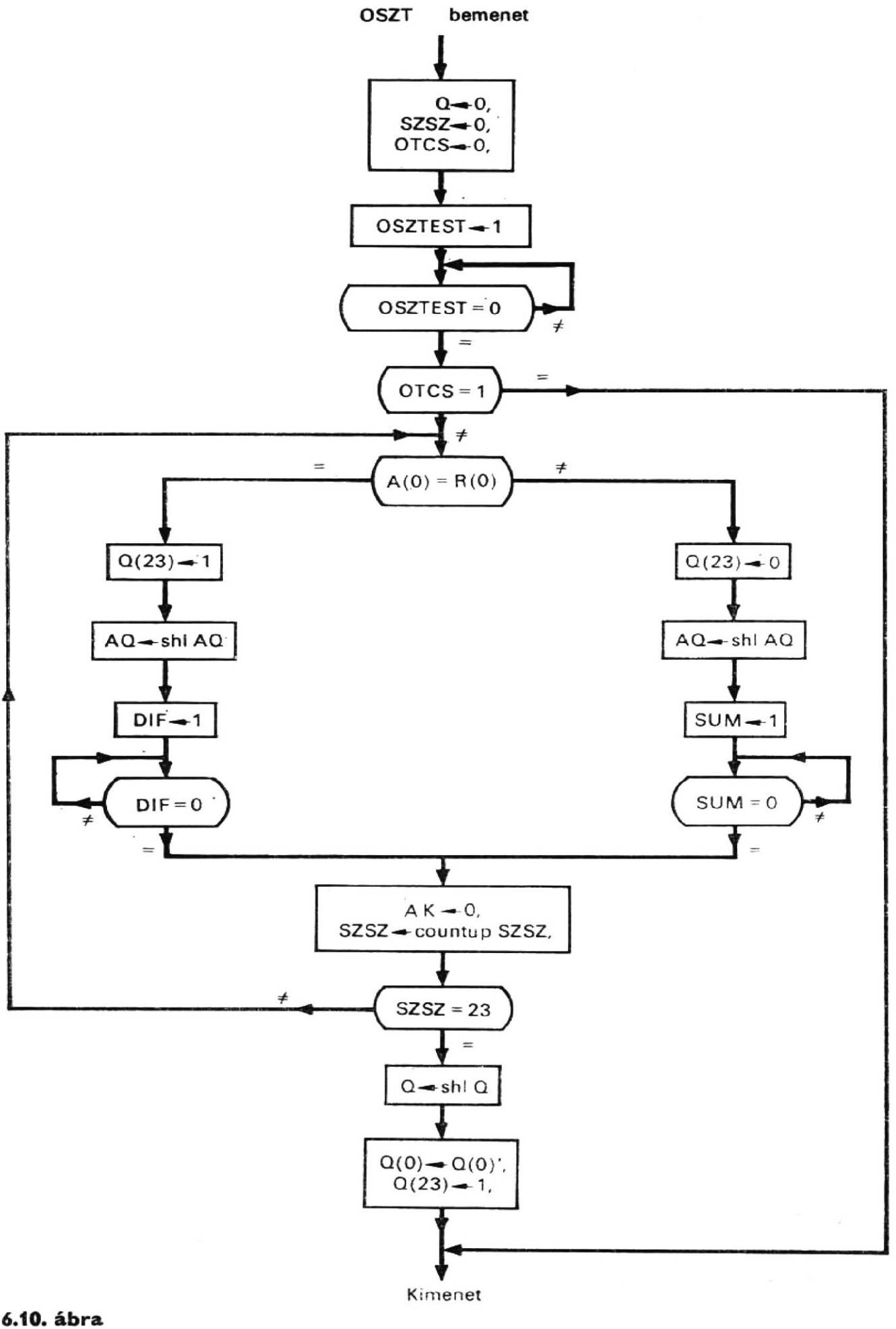
A *b*) és *c*) esetre az osztás-stop állapotokat a következő Boole-függvényekkel lehet leírni:

$$N' * A(0)' * R(0)'EA * E' \quad \text{és} \quad N * A(0)' * R(0) * EA * E'$$

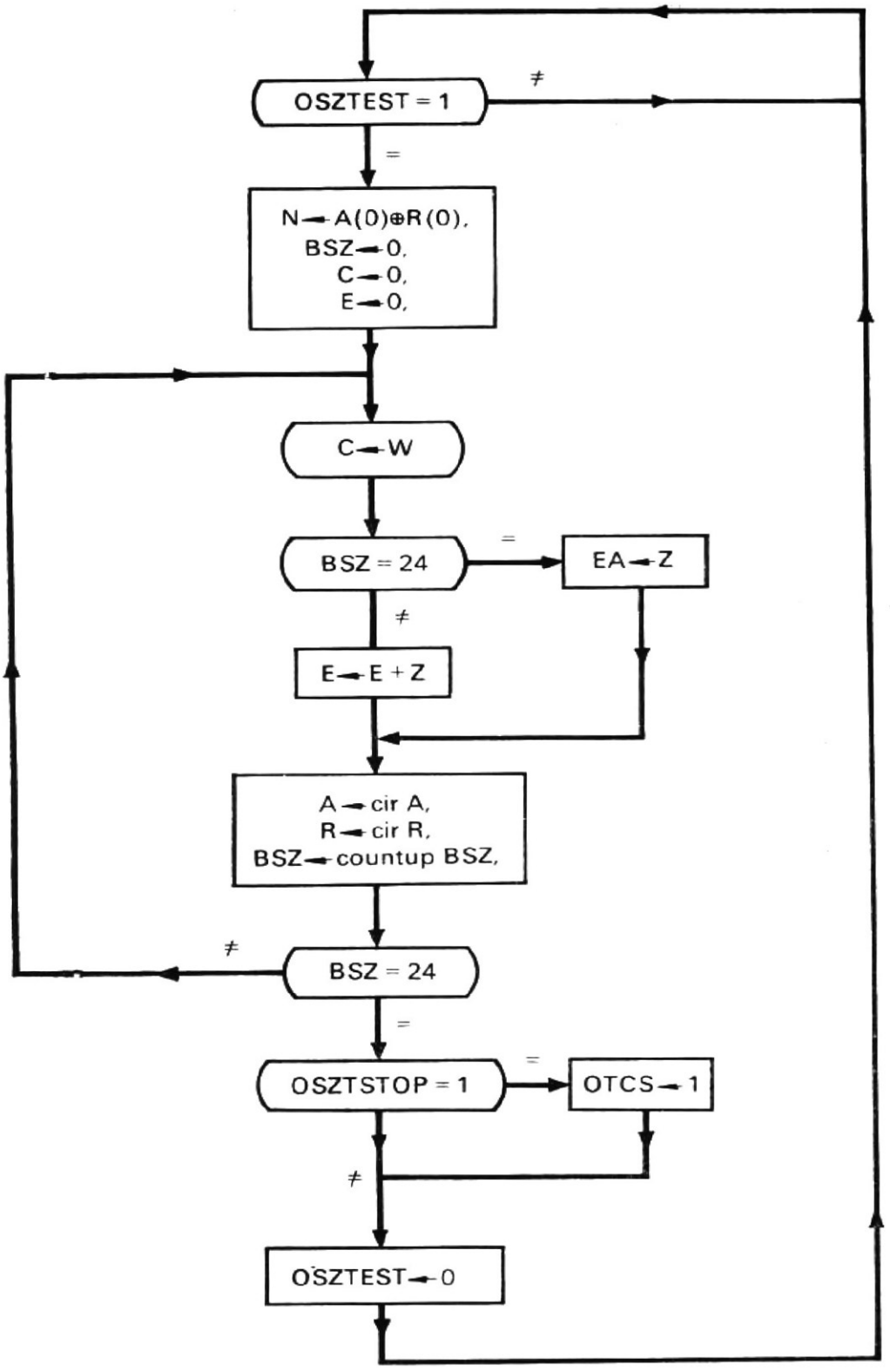
Az OSZTSTOP kimenet ily módon a fenti Boole-kifejezések logikai VAGY kapcsolata.

6.4.3. Struktúra

A visszaállítás nélküli (nonrestoring) algoritmussal végzett bináris osztás az A, R és Q regisztereket használja fel. Amint a 6.1. táblázatból látható, az osztandót és az osztót az A, ill. az R regiszter tárolja kezdetben és a Q kiindulási értéke 0. Az osztás befejezésekor a hányados a Q regiszterben, a maradék az A regiszterben jelenik meg. Az osztó az R regiszterben érintetlen marad, míg az osztandó elvész. Ha osztási túlcsordulás lép fel, ezt az OTCS regiszter jelzi ki, amelynek a tartalma ekkor 1. Amint az előzőkben említettük, az EA és az E regiszterek jelzik az előjelet és a nullaértéket az osztás-stop állapot alatt.



6.10. ábra
Folyamatábra a bináris osztási sorozathoz



6.11. ábra
Folyamatábra az osztás-stop ellenőrző részsorozathoz

6.4.4. Folyamatábra

Az osztási sorozat folyamatábráját a 6.10. ábrán láthatjuk. Ez három részt tartalmaz:

- a kezdetiérték-adást,
- az osztási túlcsordulás vizsgálatot és az
- osztást.

Amint látható, a kezdetiérték-adás csupán abból áll, hogy az OTCS, Q és SZSZ regiszterek 0 értéket kapnak. Ezután „behívják" az OSZTEST (azaz az osztás-stop ellenőrzési) részsorozatot. Ha folytatódik az osztási sorozat az OSZTEST részsorozat után, akkor az osztási rész következik.

Az osztás-stop részsorozathoz a 6.11. folyamatábra tartozik. Amint látható, ez a részsorozat várakozási hurokban van és állandóan vizsgálja az OSZTEST regisztert. Ha azt találja, hogy az OSZTEST regiszter 1-est tartalmaz, megkezdődik az OSZTEST részsorozat. Ez először is visszaállítja a BSZ, C és E regisztereket 0-ra, és 1 értéket ad az N regiszternek, ha összeadást kell végezni, vagy 0-t, ha kivonást kell végezni. Ezután megkezdődik az összeadás—kivonás hurok, amely alatt egy összeadás (vagy kivonás) hajtódik végre soros üzemmódban. Ezen összeadás (vagy kivonás) alatt az átvitelbitet (vagy átvételbitet) a C regiszter tárolja, de az összegbitet (vagy különbségbitet) elhanyagolják, mivel nincs rá szükség, kivéve a balról legelső bitpozíció bitjét, amelyet az EA regiszter tárol az OSZTSTOP kimenet előállítására. Továbbá az összeadás (vagy kivonás) alatt az összegbitnek (vagy a különbségbitnek) az E regiszter tartalmával képzett logikai VAGY kapcsolatát az E regiszter tárolja; ily módon amikor az E regiszter tartalma 0, ez azt jelenti, hogy az összeg (vagy a különbség) nagysága 0. A hurok 24-szer iterál. A hurokból való kilépés után megvizsgálják az OSZTSTOP kimenetet (terminált) túlcsordulás indikálására. Ha ilyen előfordul, az OTCS regiszter 1 értéket kap. Ekkor ez a részsorozat befejeződött, az OSZTEST regiszter 0 értéket kap, és visszatérünk az osztási sorozathoz.

A visszatérés után az OTCS regiszter megvizsgálásával kezdődik az osztási sorozat. Ha ez 1-est tartalmaz, az osztási sorozat leáll. Egyébként pedig az osztási hurok aktivizálódik. Ebben a hurokban hat funkció kerül végrehajtásra, mégpedig:

— a Q(23) bit 1 vagy 0 értéket kap,
— az AQ regisztersort léptetik balra,
— behívják a SUM-DIF részsorozatot,
— visszaállítják az AK regisztert 0-ra,
— inkrementálják, és megvizsgálják az SZSZ regisztert. Ha az SZSZ regiszter értéke még nem 23, a hurok addig folytatja az iterálást, amíg az SZSZ regiszter tartalma el nem éri a 23 értéket. Ekkor a Q regisztert egy bitpozícióval balra léptetik és elvégzik a korrekciót a hányadosbitekben. Ez a korrekció abból áll, hogy invertálják a Q(0) bitet és a Q(23) bitnek 1 értéket adnak. Az osztási sorozat ezzel véget ért.

6.5. Utasításleírások

Miután bemutattuk az összeadás, kivonás, szorzás és osztás strukturáját és sorrendi műveleteit, a következőkben CDL utasításokkal leírjuk a bináris, soros aritmetikai egység vezérlési részét és aritmetikai műveletsorozatait.

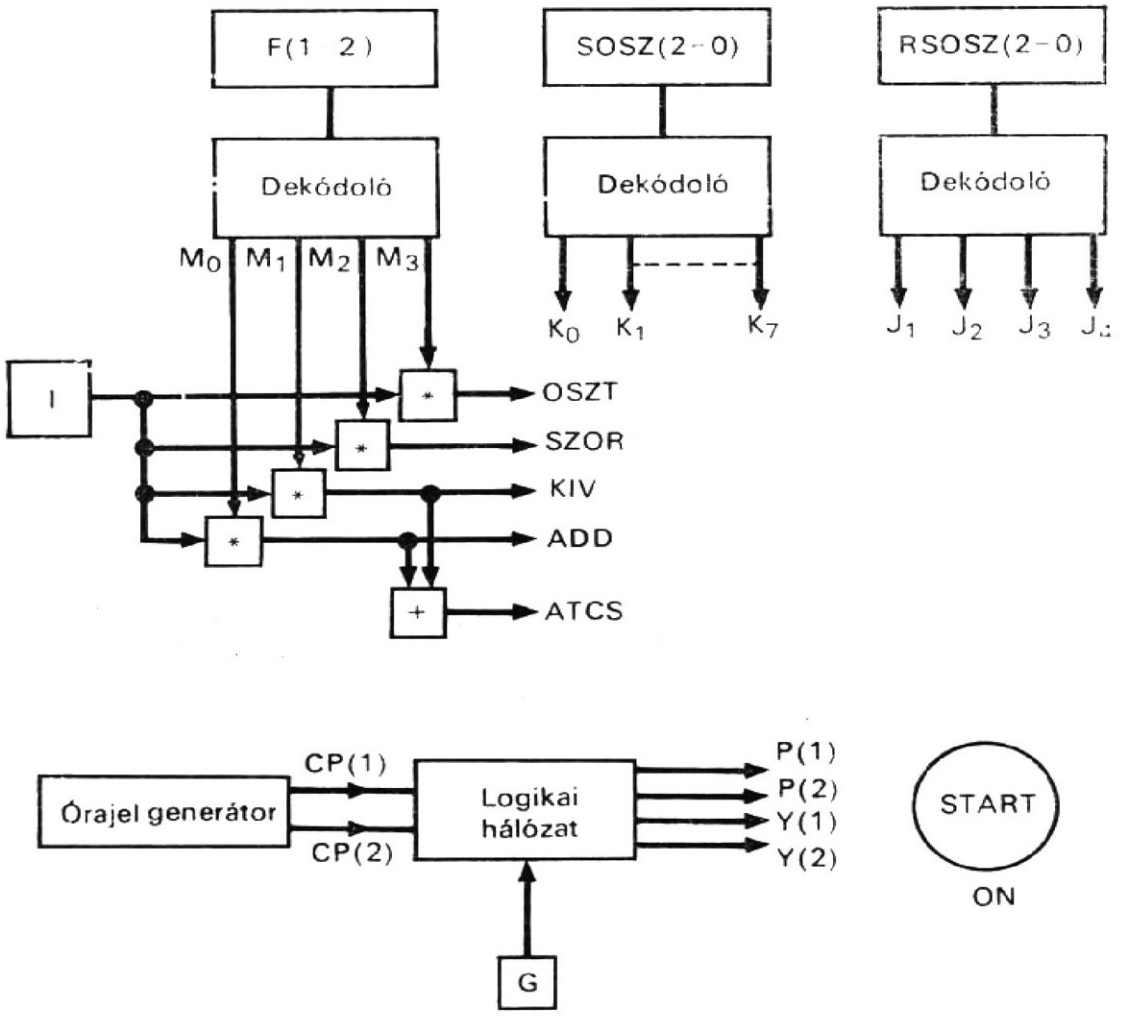
6.5.1.
Vezérlési struktúra

A bináris, soros aritmetikai egység vezérlési részéhez öt regiszter tartozik:

SOSZ, RSOSZ, F, G és I.

A SOSZ regiszter a dekódolóval együtt állítja elő a K vezérlőjeleket a négy aritmetikai művelet vezérlésére. Az RSOSZ regiszter a dekódolóval együtt állítja elő a J vezérlőjeleket a SUM-DIF és az OSZTEST részsorozatok vezérlésére. Az I regiszter jelzi az utasításlehívó ciklust (ha 1), és az utasításvégrehajtó ciklust (ha 0). Az F regiszter tárolja a műveleti kódot. Az F regiszter a dekódolóval és az I regiszterrel együtt állítja elő az aritmetikai sorozatok vezérléséhez szükséges jeleket, és egy kombinált jelet, amelyet mind az összeadásban, mind a kivonásban használhatunk. Van két, kétfázisú órajelgenerátor CP, amelynek kimenetén két kétfázisú jel, a P és az Y jelenik meg. Amikor a G regiszter 1, a P óra-, egyébként pedig az Y órajelek jelennek meg. Így a P órajelgenerátor, ill. az Y órajelgenerátor vezérelhető a G regiszter segítségével. Ha G=1 P aktív és megfordítva. A P órajeleket a négy aritmetikai művelet sorozathoz használják, míg az Y órajeleket a SUM-DIF és az OSZTEST részsorozatokhoz. Végül bevezetünk egy START kapcsolót a sorozatok indítására.

A fenti vezérlési rész struktúrája a 6.12. ábrán blokkdiagramján látható és a következő CDL deklarációs utasítások írják le:



6.12. ábra
A bináris, soros aritmetikai egység vezérlési részének struktúrája

Comment, a bináris, soros aritmetikai egység vezérlési struktúrája (6.11)

```
Register,  SOSZ(2-0),              $sorozatszámláló
           RSOSZ(2-0),             $részsorozat-számláló
           G,                      $órajelgenerátor vezérlő regiszter
           F(1-2),                 $műveletikód-regiszter
           I,                      $utasításlehívás (ha 1) és utasításvégrehajtás
                                     (ha 0)
Decoder,   M(0-3)=F,               $műveletikód-dekódoló
           K(0-7)=SOSZ,            $sorozatdekódoló
           J(1-4)=RSOSZ,           $részsorozat-dekódoló
Terminal,  SD=SUM+DIF,             $összeadás és kivonás kombinált parancs
           ADD=M(0)*I',            $összeadási parancs
           KIV=M(1)*I',            $kivonási parancs
           SZOR=M(2)*I',           $szorzási parancs
           OSZT=M(3)*I',           $osztási parancs
Clock,     CP(1-2),                $két kétfázisú órajelgenerátor
Terminal,  P(1-2)=CP(1-2)*G,       $órajel a sorozat számára
           Y(1-2)=CP(1-2)*G',      $órajel a részsorozat számára
Switch,    START(ON),
```

6.5.2. Összeadási és kivonási művelet sorozatok

Kezdetben a műveleti kód az F regiszterben van, az operandusok pedig az R, A és Q regiszterekben. A SOSZ regiszter minden P(2) óraütemben eggyel inkrementálódik. Az RSOSZ regiszter minden Y(2) óraütemben inkrementálódik. Amikor a START kapcsolót ON állásba fordítják, a SOSZ és az I regiszter 0, a G regiszter 1 értéket kap. Ezeket a mikroműveleteket a következő végrehajtási utasítások írják le:

Comment, itt kezdődnek az aritmetikai sorozatok (6.12)

```
/START(ON)/                 G←1, SOSZ←0, I←0,
/P(2)/                      SOSZ←countup SOSZ,
/(SOSZ+OSZTEST)*Y(2)/       RSOSZ←countup RSOSZ,
```

Az összeadási és kivonási sorozatok 6.5. folyamatábrájából ezeket a sorozatokat az alábbiakban írjuk le:

Comment, itt kezdődik az összeadási sorozat (6.13)

```
/ADD*K(1)*P(1)/             SUM←1, G←0, ATCS←0,
```

Comment, a sorozat most vár a SUM részsorozat befejezésére

```
/ADD*K(2)*P(1)/             IF (AK=1) THEN (ATCS←1),
/ADD*K(3)*P(1)/             I←1, SOSZ←7,
```

Comment, itt kezdődik a kivonási sorozat (6.14)

```
/KIV*K(1)*P(1)/             DIF←1, G←0, RSOSZ←0, ATCS←0,
```

Comment, a sorozat most vár a DIF részsorozat befejezésére

```
/KIV*K(2)*P(1)/             IF (AK=1) THEN (ATCS←1),
/KIV*K(3)*P(1)/             I←1, SOSZ←7,
```

A fenti utasításokban a SUM-DIF részsorozatot úgy nevezik, hogy a SUM vagy a DIF regiszter 1 értéket kap, és hogy a részsorozat a kívánt vezérlőjelekkel megindul-jon, a G és az RSOSZ regisztereket visszaállítják 0-ra. A 6.6. folyamatábrából ez a részsorozat a következőképpen írható:

Comment, itt kezdődik a SUM-DIF részsorozat (6.15)

```
/SUM*J(2)*Y(1)/      N←1,
/DIF*J(2)*Y(1)/      N←0,
/SD*J(3)*Y(1)/       BSZ←0, C←0, EA←A(0), ER←R(0), AK←0,
/SD*J(4)*Y(1)/       A←Z—A(0—22), C←W, R←cir R,
                     BSZ←countup BSZ,
/SD*J(5)*Y(1)/       IF ((BSZ=23)*(ATCSTEST=1)) THEN (AK←1),
                     IF (BSZ≠24) THEN (RSOSZ←2),
                     ELSE (IF (N=1) THEN (SUM←0),
                     ELSE (DIF←0), G←1),
```

6.5.3. Szorzási sorozat

A 6.8. folyamatábrából az alábbiakban felírjuk a szorzási sorozatot.

Comment, itt kezdődik a szorzási sorozat (6.16)

```
/SZOR*K(1)*P(1)/     E←0, A←0, SZSZ←0,
/SZOR*K(2)*P(1)/     IF (Q(23)'*E) THEN (SUM←1),
                     IF (Q(23)*E') THEN (DIF←1),
                     IF (Q(23)⊕E) THEN (RSOSZ←0, G←0),
```

Comment, a sorozat most vár a SUM vagy a DIF részsorozat befejezésére

```
/SZOR*K(3)*P(1)/     AK←0, SZSZ←countup SZSZ,
/SZOR*K(4)*P(1)/     IF (SZSZ=24)
                     THEN (Q←A(0)—Q(0—22))
                     ELSE (AQE(1—48)←AQ, SC←0),
/SZOR*K(5)*P(1)/     I←1, SOSZ←7,
```

6.5.4. Osztási sorozat

A 6.10. folyamatábrából az osztási sorozat a következőképpen írható fel:

Comment, itt kezdődik az osztási sorozat (6.17)

```
/OSZT*K(1)*P(1)/     Q←0, SZSZ←0, OTCS←0,
                     OSZTEST←1, RSOSZ←0,
```

Comment, a sorozat vár az OSZTEST részsorozat befejezésére

```
/OSZT*K(2)*P(1)/     IF (OTCS=1) THEN (SOSZ←6),
/OSZT*K(3)*P(1)/     IF (A(0)=R(0))
                     THEN (Q(23)←1) ELSE (Q(23)←0)),
```

```
/OSZT * K(4) * P(1)/      IF (A(0)=R(0))
                          THEN (DIF←1) ELSE (SUM←1),
                          AQ←sh1 AQ, G←0, RSOSZ←0,
```

Comment, a sorozat vár a SUM vagy a DIF részsorozat befejezésére

```
/OSZT * K(5) * P(1)/      AK←0, SZSZ←countup SZSZ,
/OSZT * K(6) * P(1)/      IF (SZSZ≠23)
                          THEN (SOSZ←1) ELSE (Q←sh1 Q),
/OSZT * K(7) * P(1)/      Q(0)←Q(0)', Q(23)←1, I←1,
```

A SUM-DIF részsorozat leírását az előbbiekben mutattuk be. A 6.11. folyamatábrából az OSZTEST részsorozat a következőképpen írható le:

Comment, itt kezdődik az OSZTEST részsorozat (6.18)

```
/OSZTEST * J(2) * Y(1)/   N←A(0)⊕R(0), BSZ←0, C←0,
/OSZTEST * J(3) * Y(1)/   C←W, EA←Z, A←cir A,
                          R←cir R, BSZ←countup BSZ,
/OSZTEST * J(4) * Y(1)/   IF (BSZ≠24)
                          THEN (RSOSZ←1, E←E+Z)
                          ELSE (IF (OSZTSTOP=1),
                          THEN (OTCS←1, OSZTEST←0, G←1),
```

6.6. Decimális aritmetikai egység

A decimális aritmetikai egységet a bináris, soros aritmetikai egységhez hasonlóan lehet felépíteni. A következőkben bemutatjuk a működésmódot, a decimális összeadó/kivonóegységet, a struktúrát, és egy decimális aritmetikai egység részletes működését.

6.6.1. Binárisan kódolt decimális számok

A decimális számokat rendszerint úgy ábrázolják, hogy a decimális számjegyeik bináris számokként vannak kódolva. Például a decimális számjegyeket a következőképpen lehet ábrázolni: 8–4–2–1 kóddal, 2–4–1–2 kóddal, 5–1–1–1–1 kóddal, 3-többletes kóddal, vagy 2-az-5-ből kóddal. A decimális számokat bármelyik ábrázolásban *binárisan kódolt decimális számoknak* (vagy egyszerűen BCD számoknak) nevezzük. A BCD számok részletesebb tárgyalása megtalálható másutt [4]. Az itt bemutatásra kerülő decimális aritmetikai egységhez a 8–4–2–1 kódot választottuk. Ez a négybites kód az első tíz bináris számot használja fel a 0,1,2,...,9 tíz számjegy ábrázolására, míg a maradék hat bináris szám nincs kihasználva.

Egy decimális szám előjeles vagy előjel nélküli lehet. Az előjeles decimális számnak előjelbitje is van. Az előjelbit rendszerint a szám legnagyobb helyiértékű számjegy pozíciója mellett helyezkedik el, és az értékének 0-t vagy 1-et választanak attól függően, hogy az előjel pozitív, ill. negatív. Az előjeles bináris számoknak három ábrázolása van:

1. Az előjellel és abszolút értékkel történő ábrázolás.
2. Az előjeles, 10-es komplemens ábrázolás.
3. Az előjeles, 9-es komplemens ábrázolás.

Ha a decimális szám pozitív, mind a három ábrázolásban az előjelen kívüli számjegyek a decimális szám értékeit ábrázolják. Ha viszont a szám negatív, a számjegyek mást és mást ábrázolnak. Az előjellel és abszolút értékkel történő ábrázolásban az előjelen kívüli számjegyek ugyancsak az értéket reprezentálják, míg a másik két ábrázolásban ezek az érték 10-es komplemensét vagy 9-es komplemensét reprezentálják. Ezek az ábrázolási módok hasonlóak az előjellel és abszolút értékkel, ill. komplemens kóddal történő bináris számábrázoláshoz.

A bináris szám előjeles 2-es komplemens ábrázolásához hasonlóan a decimális szám előjeles 10-es komplemens ábrázolása a decimális szám aktuális értékét adja, ha az előjel súlyozási tényezőjét tekintjük. Például vegyük az 1523 (vagy −477) decimális számot az előjeles 10-es komplemens ábrázolásban. Ha az előjel-számjegy értékét −1000-nek vesszük +1000 helyett, akkor:

$$\begin{aligned} 1523 &= -1\times10^3+5\times10^2+2\times10^1+3\times10^0 \\ &= -1000+523 \\ &= -477, \end{aligned}$$

ahol a −477 az 1523 aktuális értéke. Az előjel-számjegynek negatív számjegyként történő kezelése nem hat ki a pozitív decimális számokra, ugyanis ha az előjel (számjegy) pozitív, az értéke nulla.

Az előzőkben említett 2–4–2–1 kód vagy a 3-többletes kód egyes decimális jegyei önkomplementálóak, és így az eredeti decimális számnak a 9-es komplemense a számjegyek 9-es komplemens képzésével nyerhető. Az ilyen kódokat *önkomplementáló kódoknak* nevezik. Az önkomplementáló kódban levő decimális számnak közvetlenül előállítható a 9-es komplemense.

A decimális összeadás és kivonás módszerei kissé hasonlók a bináris számokéhoz. Azonban eltérés is van, ha a BCD számok decimális számjegyeket reprezentálnak. Illusztráljuk az eltérést egy példán, ahol két, 8–4–2–1 kódban kifejezett decimális számot (ezek legyenek: 795 és 683) adunk össze, az alábbiak szerint.

Decimális	Százasok		Tízesek	Egyesek
	1 ←			
795	0111		1001	0101
+ 683	+0110	átvitel	+1000	+0011
1478	1110	↑	10001	1000
	+0110		+0110	
	10100/4/		0111/7/	1000/8/

↑ (átvitel az ezresek pozíciójába)

Az 1000 eredmény az egyesek helyiértékén helyes, viszont az összeadás során kapott 0001 a tízesek pozíciójában helytelen. Nemcsak az 1110 értéke helytelen a százasok pozíciójában az összeadás után, hanem az átvitel is hiányzik. Viszont, ha a 0110 bináris számot hozzáadjuk mind a tízesek, mind a százasok pozíciójában kapott összeghez, az eredmény most már minden helyiértéken helyes lesz és az átvitel is megjelenik a százasok pozíciójában. Ezért a decimális szám ábrázolásának megválasztásával együtt ki kell fejleszteni egy korrekciós algoritmust is, hogy két decimális számokat helyesen lehessen összeadni vagy kivonni, és az átvitel (átvétel) is helyes legyen.

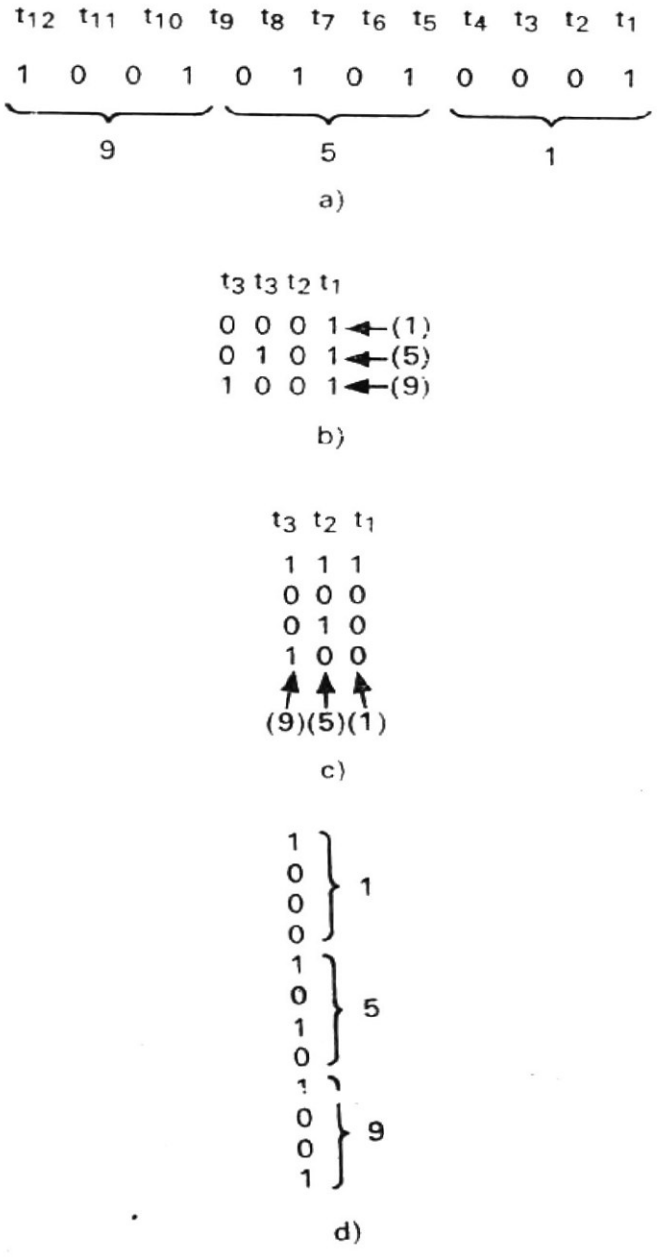
Összegezve, a decimális aritmetikai egységhez a decimális számot előjellel és abszolút értékkel ábrázoltuk. A decimális számot a 8–4–2–1 kóddal reprezentáljuk. A számok nyolc decimális számjegyű egész számok, amelyekhez egy előjelbit járul.

6.6.2. Működésmódok

A binárisan kódolt decimális számot 1-ek és 0-ák egyszeres vagy többszörösi sorozataival lehet ábrázolni. Négy ilyen ábrázolási mód van:

1. A soros számjegy—soros bit ábrázolás.
2. A párhuzamos számjegy—soros bit ábrázolás.
3. A soros számjegy—párhuzamos bit ábrázolás.
4. A párhuzamos számjegy—párhuzamos bit ábrázolás.

Példaként nézzük meg a 951 decimális számot, 8–4–2–1 kódban ez: 1001, 0101, 0001 lesz. Ennek a számnak négyféle ábrázolásmódja a 6.13. ábrán látható, ahol t_i az idő, amely úgy nő, ahogy az i index növekszik. A 6.13*a* ábrán valamennyi számjegy és valamennyi bit időben egymás után jelenik meg; ez a soros számjegy—soros bit ábrázolás. Ha a számjegyek párhuzamosan jelennek meg és a számjegyeket képező bitek sorban, amint a 6.13*b* ábrán látható, akkor ez a párhuzamos számjegy—soros bit ábrázolás. Ha a számjegyek sorosan fordulnak elő, míg a bitek párhuzamosan, amint a



6.13. ábra
Binárisan kódolt decimális szám négyféle ábrázolása
a) **soros számjegy—soros bit ábrázolás;**
b) **párhuzamos számjegy—soros bit ábrázolás;**
c) **soros számjegy—párhuzamos bit ábrázolás;**
d) **párhuzamos számjegy—párhuzamos bit ábrázolás**

6.13*c* ábra mutatja, akkor ez a soros számjegy—párhuzamos bit reprezentáció. A 6.13*d* ábrán valamennyi számjegy és valamennyi bit párhuzamosan jelenik meg; ez a párhuzamos számjegy—párhuzamos bit megjelenítés.

Két decimális szám összeadását olyan decimális összeadó felhasználásával végezhetjük el, amely vagy egy, vagy több decimális összeadót tartalmaz. A decimális összeadó lehet soros, ha a két decimális számjegyet sorosan adják össze (azaz decimális számjegyenként), vagy párhuzamos, ha párhuzamosan adják össze őket (vagyis valamennyi bitet egyszerre). Ezeket az összeadókat *soros decimális összeadónak* (SDA) vagy *párhuzamos decimális összeadónak* (PDA) nevezik. Az a mód, ahogy két decimális számot összeadnak, függ attól, hogy hogyan ábrázolták a BCD számot. Négy működésmód lehetséges, a négyféle BCD számábrázolásnak megfelelően. Ha a BCD számot soros számjegy—soros bit alakban jelenítik meg, az összeadáshoz SDA-t kell használni, és ekkor *soros üzemmódú* a működése. Ha a BCD szám ábrázolása párhuzamos számjegy—soros bit, az összeadáshoz annyi SDA kell, ahány számjegy van a decimális számban és ezt a műveletet *párhuzamos számjegy üzemmódnak* nevezik. Ha a BCD számot soros számjegy—párhuzamos bit reprezentálja, az összeadáshoz PDA szükséges, és *soros számjegy üzemmódú* a működése. Ha a BCD szám párhuzamos számjegy—párhuzamos bit ábrázolásban jelenik meg, az összeadáshoz annyi PDA szükséges, amennyi a számjegyek száma a decimális számban és ezt *párhuzamos üzemmódú működésnek* nevezik.

Hasonlóképpen négy működésmód lehetséges egyik decimális számnak a másikból való kivonására is. Ezekhez a működésmódokhoz egy vagy több, soros decimális kivonó (SDK), vagy egy vagy több, párhuzamos decimális kivonó (PDK) szükséges.

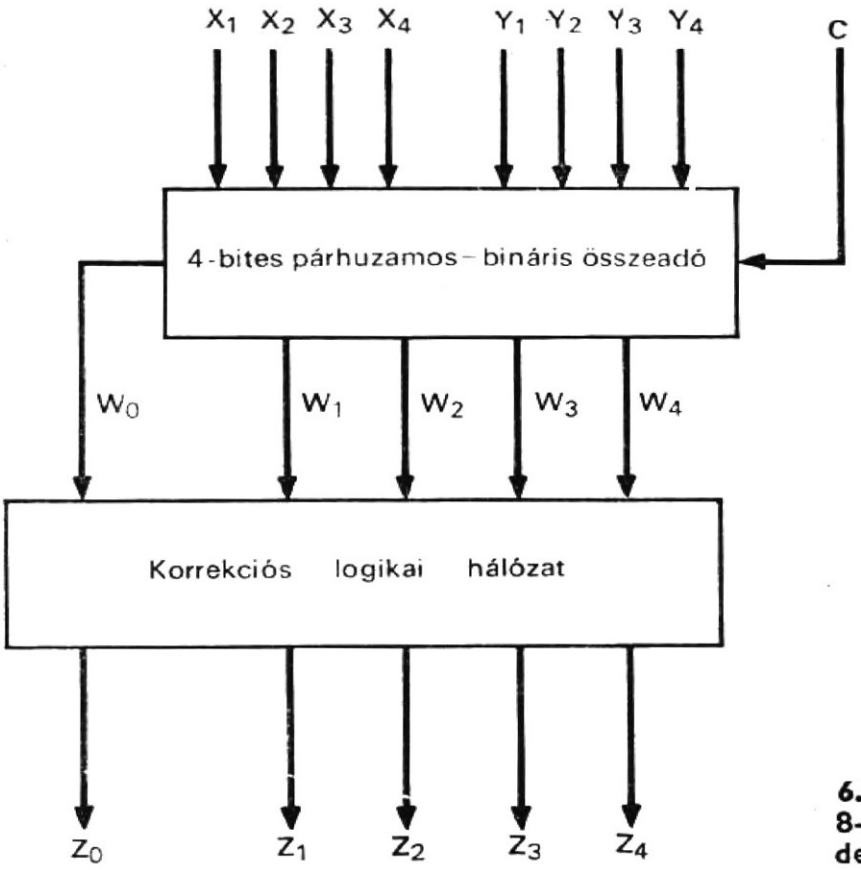
Az összeadót és a kivonót összeadó—kivonóvá lehet egyesíteni. Ebben az esetben *soros decimális összeadó—kivonóról* (SDAK) és *párhuzamos decimális összeadó/kivonóról* (PDAK) beszélünk.

A decimális aritmetikai egységhez a BCD számokat soros számjegy—párhuzamos bit ábrázolásban választottuk meg. Ezért PDA, PDK vagy PDAK szükséges; ami az adott választás és a felhasznált aritmetikai algoritmustól függ.

6.6.3. Decimális összeadók és kivonók

A decimális összeadó összeadja a két összeadandó számjegyet és a bemeneti átvitelt, kimenetén létrehozza az összeg-számjegyet és a kimeneti átvitelbitet. A továbbiakban vizsgált decimális összeadók és kivonók a 8–4–2–1 kódú, binárisan kódolt decimális számjegyeket használják. Egy párhuzamos decimális összeadót (PDA) láthatunk a 6.14. ábra blokkdiagramján. Ennek három bemenete van: az X(1—4), ill. az Y(1—4) 4-bites összeadandó, és a C egybites bemeneti átvitel. Két kimenete van: a Z(1—4) 4-bites összeg és a Z(0) egybites kimeneti átvitel.

A 8–4–2–1 kód alkalmazása lehetővé teszi, hogy bináris összeadót használjunk a decimális összeadó részeként. A 6.14. ábrán látható PDA egy 4-bites párhuzamos összeadót és egy korrekciós logikai hálózatot tartalmaz. A bináris összeadóból érkező W(0—4) kimenetek adják a bemenetek bináris összegét. Ennek a bináris összegnek 19 lehetséges értéke van, ezeket adtuk meg a 6.4. táblázat bal oldali oszlopában, itt az átvitel a vesszőtől balra helyezkedik el. A jobb oldali oszlopban levő értékek a Z(0—4) helyes összegek. A korrekciós logikai hálózat végzi a bal oldali oszlopban levő összegnek a jobb oldali oszlopban található, helyes összeggé történő átalakítását. A 6.4. táblázat figyelmes tanulmányozásával felfedezhetünk és megfogalmazhatunk egy egyszerű korrekciós szabályt. Az első 10 esetben az összeg (0...9) korrekt; tehát



6.14. ábra
8-4-2-1 kódot alkalmazó, párhuzamos, decimális összeadó blokkdiagramja

6.4. táblázat

Két decimális számjegy nemkorrigált és korrigált összegei, 8-4-2-1 kódban

Nemkorrigált összeg W(0—4)	Korrigált összeg Z(0—4)
0,0000	0,0000 = 0
0,0001	0,0001 = 1
0,0010	0,0010 = 2
0,0011	0,0011 = 3
0,0100	0,0100 = 4
0,0101	0,0101 = 5
0,0110	0,0110 = 6
0,0111	0,0111 = 7
0,1000	0,1000 = 8
0,1001	0,1001 = 9
0,1010	1,0000 = 10
0,1001	1,0001 = 11
0,1100	1,0010 = 12
0,1101	1,0011 = 13
0,1110	1,0100 = 14
0,1111	1,0101 = 15
1,0000	1,0110 = 16
1,0001	1,0111 = 17
1,0010	1,1000 = 18
1,0011	1,1001 = 19

helyesbítésre nincs szükség. A további 10 összeg (10...19) mindegyikénél a nem korrigált összeg 6-tal való növelése szükséges; ezért a bal oldali oszlopban a W(0—4) összeget 6-tal megnöveljük, hogy megkapjuk a jobb oldali oszlopban a helyes Z(0—4) kódot.

A fenti, 8–4–2–1 kódot felhasználó PDA-t most egy „decadd" operátorként definiáljuk a következő CDL utasítások segítségével.

Comment, 8–4–2–1 kódot felhasználó, párhuzamos decimális összeadó specifikálása

Operator, $Z(0-4)=X(1-4)$ decadd $Y(1-4)-C$ (6.19)

Comment, C a bemeneti átvitel és W(0) a kimeneti átvitel

Terminal,
$$CB(4)=C,$$
$$CB(0-3)=X(1-4)*Y(1-4)+Y(1-4)*$$
$$*CB(1-4)+CB(1-4)*X(1-4),$$
$$W(0)=CB(0),$$
$$W(1-4)=X(1-4)\oplus Y(1-4)\oplus CB(1-4),$$

Comment, W(1—4) a 4-bites párhuzamos összeadó összegkimenete

/begin/ IF $(W=0+1+\ldots+9)$ THEN $(Z=W)$,
IF $(W=10+11+\ldots+19)$ THEN $(Z=W$ add 6),
end of operator

A fenti terminálutasítás a 4-bites párhuzamos bináris összeadót írja le, amelynek a W(0—4) kimeneti kapcsok (terminálok) a kimenetei. A két feltételes mikroutasítás a szükséges korrekciót írja le.

A párhuzamos decimális kivonót hasonlóképpen lehet definiálni. Ez ismét egy 4-bites bináris kivonót és egy korrekciós logikai hálózatot tartalmaz. A bináris kivonó bemenetei a következők: az X(1—4) 4-bites kisebbítendő, az Y(1—4) 4-bites kivonandó és a C bemeneti átvétel. A bináris kivonóról érkező kimenetek a W(0—4) terminálok. A W(0) kimenet a kimeneti átvétel és a W(1—4) kapcsok képezik a különbséget. A PDK-ról érkező korrigált kimenetek jele Z(0—4), ahol a Z(0) a kimeneti átvétel és Z(1—4) a különbség. A 20 nemhelyesbített átvétel és a W(0—4) különbség, valamint a hozzájuk tartozó helyesbített átvétel és a Z(0—4) különbség a 6.5. táblázatban található, ahol a zárójelbe foglalt átvétel a vesszőtől balra helyezkedik el. A korrekciós logikai hálózat a bal oldali oszlopban levő különbséget átalakítja a jobb oldali oszlopban előírt értékűvé. Bárhol is jelenik meg átvétel, ott a számjegy negatív és 10-es komplemens alakban van. A 6.5. táblázat tanulmányozása ismét egy egyszerű korrekciós szabály megfogalmazásához vezet. Az első 10 különbség (0...9) helyes, tehát nincs szükség korrekcióra. A további 10 különbség mindegyikénél (—1-től —10-ig) 6-tal való csökkentés szükséges; ezért a bal oldali oszlopban levő W(0—4) különbséget 6-tal csökkentik, hogy megkapják a jobb oldali oszlopban levő helyes Z(0—4) kódot.

A fenti PDK-t, amely 8–4–2–1 kódot használ, a „deckiv" operátorként definiáljuk az alábbiakban, CDL utasítások segítségével.

Comment, 8–4–2–1 kódot felhasználó, párhuzamos decimális-kivonó specifikálása

Operator, $Z(0-4)=X(1-4)$ deckiv $Y(1-4)-C$, (6.20)

Comment, C a bemeneti átvétel és W(0) a kimeneti átvétel

Terminal,
$$CB(4)=C,$$
$$CB(0-3)=X(1-4)'*Y(1-4)+Y(1-4)*$$
$$*CB(1-4)+CB(1-4)*X(1-4)',$$
$$W(0)=CB(0),$$
$$W(1-4)=X(1-4)\oplus Y(1-4)\oplus CB(1-4),$$

6.5. táblázat

Két decimális számjegy nemkorrigált és korrigált különbsége, 8-4-2-1 kódban*

Nem korrigált különbség W	Korrigált különbség Z
0,1001	0,1001 = 9
0,1000	0,1000 = 8
0,0111	0,0111 = 7
0,0110	0,0110 = 6
0,0101	0,0101 = 5
0,0100	0,0100 = 4
0,0011	0,0011 = 3
0,0010	0,0010 = 2
0,0001	0,0001 = 1
0,0000	0,0000 = 0
(1),1111	(1),1001 = −1
(1),1110	(1),1000 = −2
(1),1101	(1),0111 = −3
(1),1100	(1),0110 = −4
(1),1011	(1),0101 = −5
(1),1010	(1),0100 = −6
(1),1001	(1),0011 = −7
(1),1000	(1),0010 = −8
(1),0111	(1),0001 = −9
(1),0110	(1),0000 = −10

* A zárójelben levő 1-es átvitelt jelöl; a számjegy 10-es komplemensben van.

Comment, W(1—4) a 4-bites párhuzamos bináris kivonó különbségkimenete

/begin/ IF (W(0)=0) THEN (Z=W),
IF (W(0)=1) THEN (Z=W sub 6),
end operator

A fenti terminálutasítás írja le a 4-bites párhuzamos bináris kivonót, ahol W(0—4) ennek a kivonónak a kimenete. A két feltételes mikroutasítás a korrekciót tartalmazza.

A fenti PDA-t és PDK-t egyetlen PDAK-vá lehet egyesíteni. Ebben az esetben szükség van az N bemenetre. A PDAK összeadóként működik, ha N=1, és kivonóként ha N=0. Ezt a 8–4–2–1 kódot alkalmazó PDAK-t a következőkben egy „decaddkiv" operátorként definiáljuk, CDL utasítások segítségével.

Comment, 8–4–2–1 kódot alkalmazó, párhuzamos, decimális összeadó—kivonó specifikálása (6.21)

Operator, Z(0—4)=X(1—4) decaddkiv Y(1—4)—C—N,

Comment, ha N=1, összeadóként, ha N=0, kivonóként működik

Terminal, CB(4)=C,
CB(0—3)=(N ⊙ X(1—4)) * Y(1—4)+(N ⊙ X(1—4)) * CB(1—4)+
+Y(1—4) * CB(1—4),

```
W(0)=CB(0),
W(1-4)=X(1-4)⊕Y(1-4)⊕CB(1-4),
```

Comment, ha az N=1, akkor a C a bemeneti átvitel, ha N=0, akkor C a bemeneti átvétel

```
/begin/   IF (N*(W=0+1+...+9)+N'*W(0)') THEN (Z=W),
          IF (N*(W=10+11+...+19)) THEN (Z=W add 6),
          IF (N'*W(0)) THEN (Z=W sub 6),
          end of operator
```

6.6.4.
Decimális, soros üzemű aritmetikai egység

Soros számjegy—párhuzamos bit ábrázolású decimális számokkal történő műveletvégzéshez használható az az aritmetikai egység, amelynek blokkdiagramja a 6.15. ábrán látható. Itt az előzőkben definiált PDAK-t az egybites C és N regiszterrel együtt használják. Ez az aritmetikai egység a 6.2. ábra blokkdiagramján bemutatott bináris, soros aritmetikai egységhez hasonló működésű.

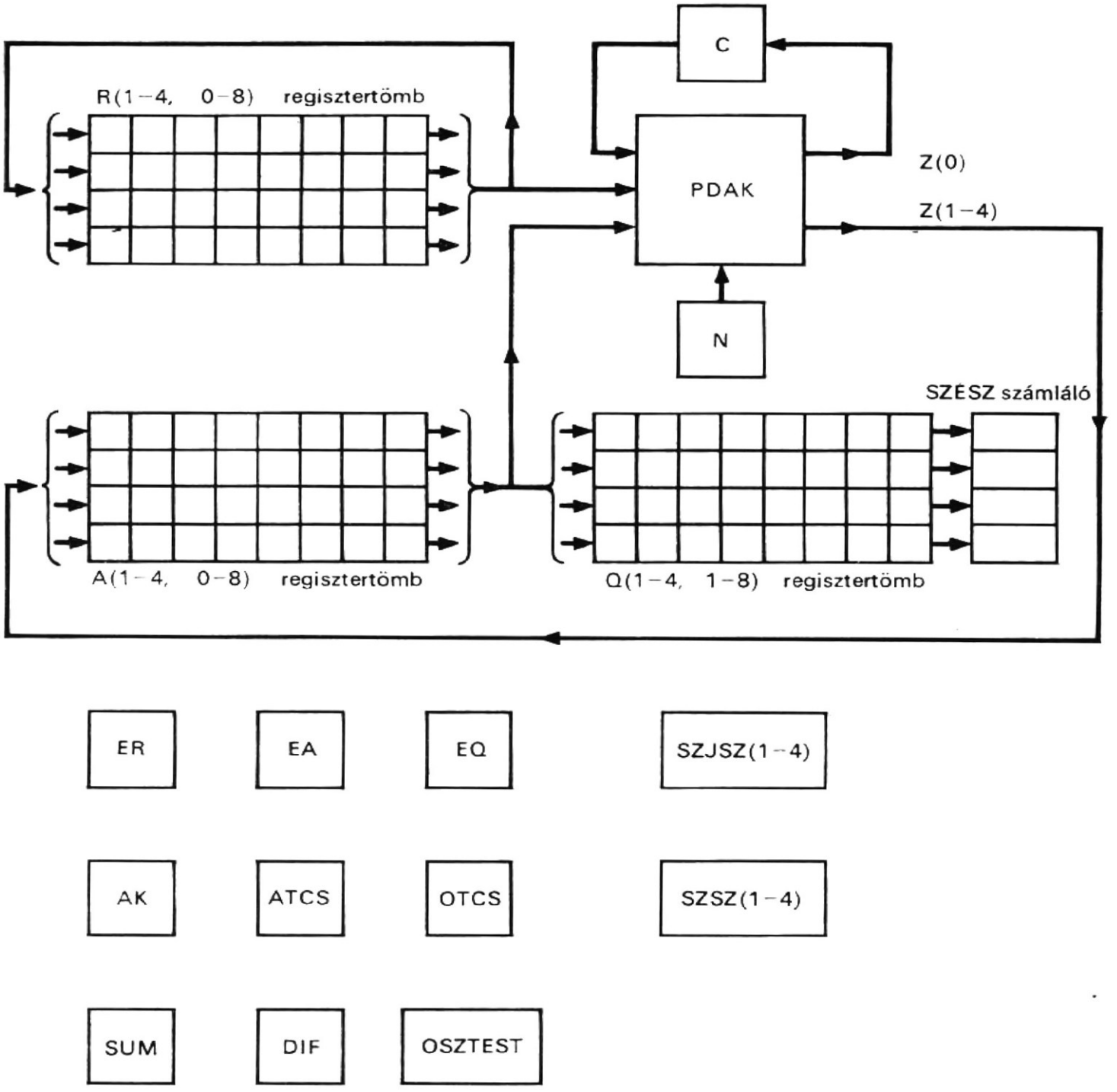
Amint a 6.15. ábrán látható, három regisztertömböt alkalmazunk: az A(1—4, 0—8), a Q(1—4, 1—8) és az R(1—4, 0,8), amelyek közül az első a decimális akkumulátor, a második a szorzó—hányadosregiszter és a harmadik a tár tartalomregisztere. Az A regiszter tömb elemeit az indexei határozzák meg, a 6.16 ábrán látható módon. Az A regisztertömb kilenc számjegyet tud tárolni, viszont a választott decimális számnak csak nyolc decimális számjegye van, az ezeken kívüli, extra számjegy tárolására a szorzáskor és osztáskor lesz szükségünk. Bár az R regisztertömbben nincs szükség az extra számjegy tárolására, a kényelem kedvéért itt is bevezettük. Az egybites ER, EA és EQ regiszterek a decimális számok előjelét tárolják az R, Q, ill. az A regisztertömbök sorrendjében. Ennek a három regisztertömbnek a funkciói az összeadás, kivonás, szorzás és osztás folyamán ugyanazok, mint a 6.1. táblázatban láthatóké, azzal az eltéréssel, hogy osztáskor az osztandónak 16 számjegye van és az A és a Q regisztertömb tárolja.

Továbbá van nyolc darab egybites regiszter: SUM, DIF, OSZTEST, AK, ATCS, OTCS, C és N. Ezek a regiszterek ugyanazt a célt szolgálják, mint amit a 6.2. ábra blokkdiagramján és a (6.7) utasításban már bemutattunk. Van három számláló: SZJSZ, SZÉSZ és SZSZ. Az SZJSZ számjegyszámlálót összeadáskor és kivonáskor használják, míg a SZÉSZ számjegyérték-számlálót és az SZSZ szószámlálót szorzáskor és osztáskor alkalmazzák.

Ennek a decimális aritmetikai egységnek a struktúráját a következő CDL utasítások írják le:

Comment, decimális aritmetikai egység struktúrája (6.22)

```
Array-register,   A(1-4, 0-8),    $akkumulátortömb
                  Q(1-4, 1-8),    $szorzó-hányados regisztertömb
                  R(1-4, 0-8),    $tartalom regisztertömb
Register,         EA,             $az A-ban levő szám előjelét tárolja
                  EQ,             $a Q-ban levő szám előjelét tárolja
                  ER,             $az R-ben levő szám előjelét tárolja
Array-casregister, AQ(0-16)=A-Q,
Register,         SUM,            $vezérlőregiszter az összeadási részsorozathoz
                  DIF,            $vezérlőregiszter a kivonási részsorozathoz
```



6.15. ábra
Párhuzamos számjegy—soros bit ábrázolást alkalmazó decimális aritmetikai egység struktúrája

					Kódsúlyozási tényező
A(1, 0)	A(1, 1)	A(1, 2)		A(1, 8)	8
A(2, 0)	A(2, 1)	A(2, 2)		A(2, 8)	4
A(3, 0)	A(3, 1)	A(3, 2)		A(3, 8)	2
A(4, 0)	A(4, 1)	A(4, 2)		A(4, 8)	1

6.16. ábra
Az A(1-4, 0-8) regisztertömb elemeinek jelölését bemutató blokkdiagram

	OSZTEST,	$vezérlőregiszter az osztás-stop ellenőrző részsorozathoz
	AK,	$átmeneti regiszter az átvitelhez vagy átvételhez
	ATCS,	$összeadási túlcsordulás-indikátor
	OTCS,	$osztási túlcsordulás-indikátor
	C,	$átvitelregiszter
	N,	$add-kiv vezérlésregiszter (összeadás esetén N=1)
Register,	SZÉSZ(1—4),	$számjegyérték-számláló
	SZJSZ(1—4),	$számjegyszámláló
	SZSZ(1—4),	$szószámláló

Comment, a párhuzamos, decimális számjegy összeadó—kivonó kapocspárjai (termináljai)

Terminal, Z(0—4)=A(,8) decaddkiv R(,8)—C—N,

A fenti terminálutasítás leírja a PDAK bemeneteit és kimeneteit, valamint magát a PDAK-t is. A (6.21) utasításban leírt „decaddkiv" operátor ezen struktúra leírásának egyik része.

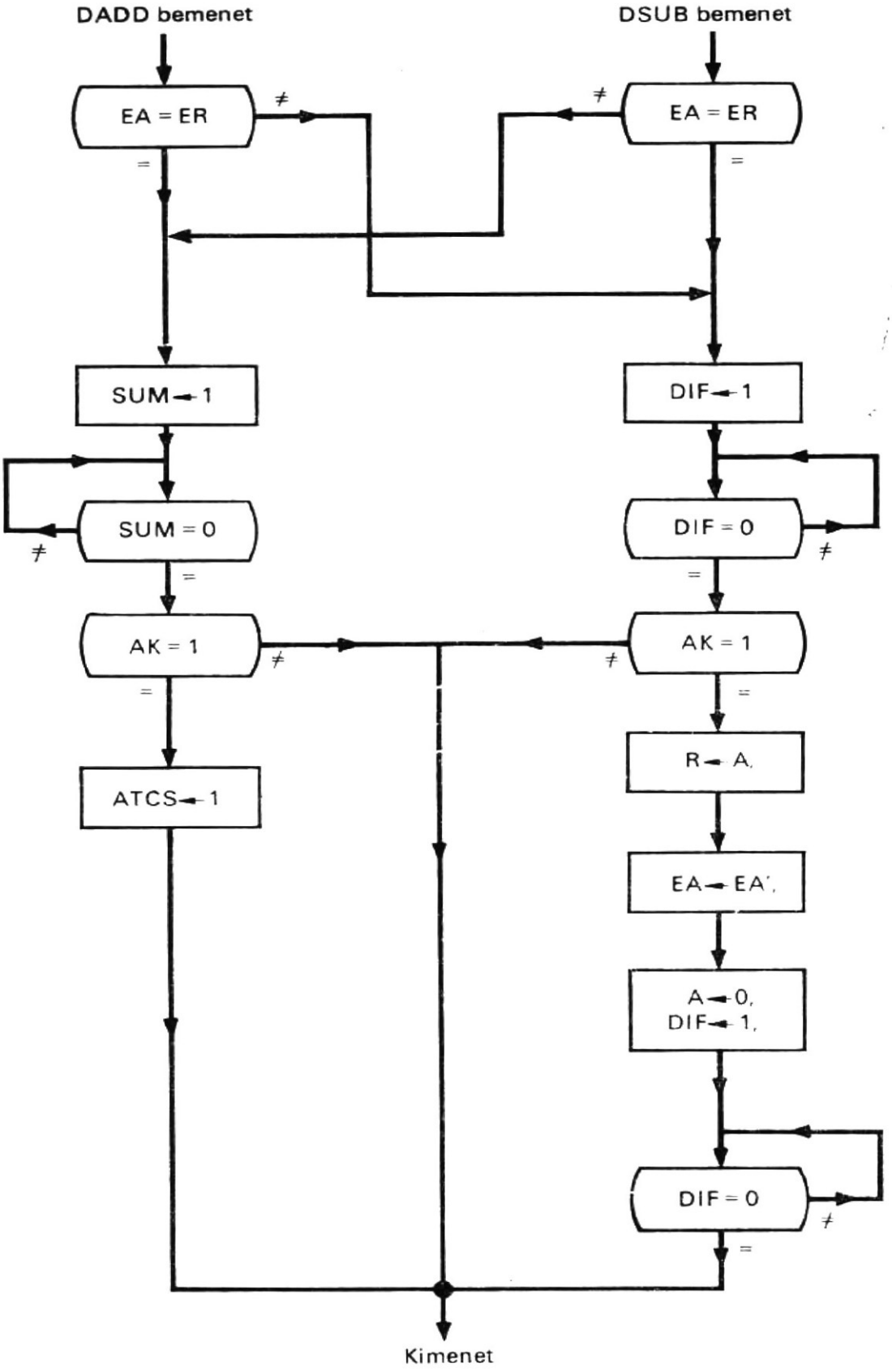
6.6.5. Decimális összeadás és kivonás

Ebben a decimális aritmetikai egységben az összeadáshoz és kivonáshoz a közvetlen összeadás és közvetlen kivonás módszereit használjuk fel [4]. Kezdetben az egyik összeadandót és a kisebbítendőt az A regisztertömb, a másik összeadandót és kivonandót az R regisztertömb tárolja. Az összeadás vagy a kivonás után az összeget, ill. a különbséget az A regisztertömb tárolja, a kezdetben A-ban levő összeadandó, ill. a kisebbítendő elvész. (Az R regiszterben levő összeadandó esetleg szintén elvész.)

A decimális összeadási és kivonási sorozatot a 6.17. és 6.18. folyamatábrák mutatják. A 6.18. folyamatábrán a SUM-DIF részsorozat látható, amelyet „behívnak" az összeadási és kivonási sorozat alatt, ha a SUM vagy a DIF regiszter 1 értéket kap.

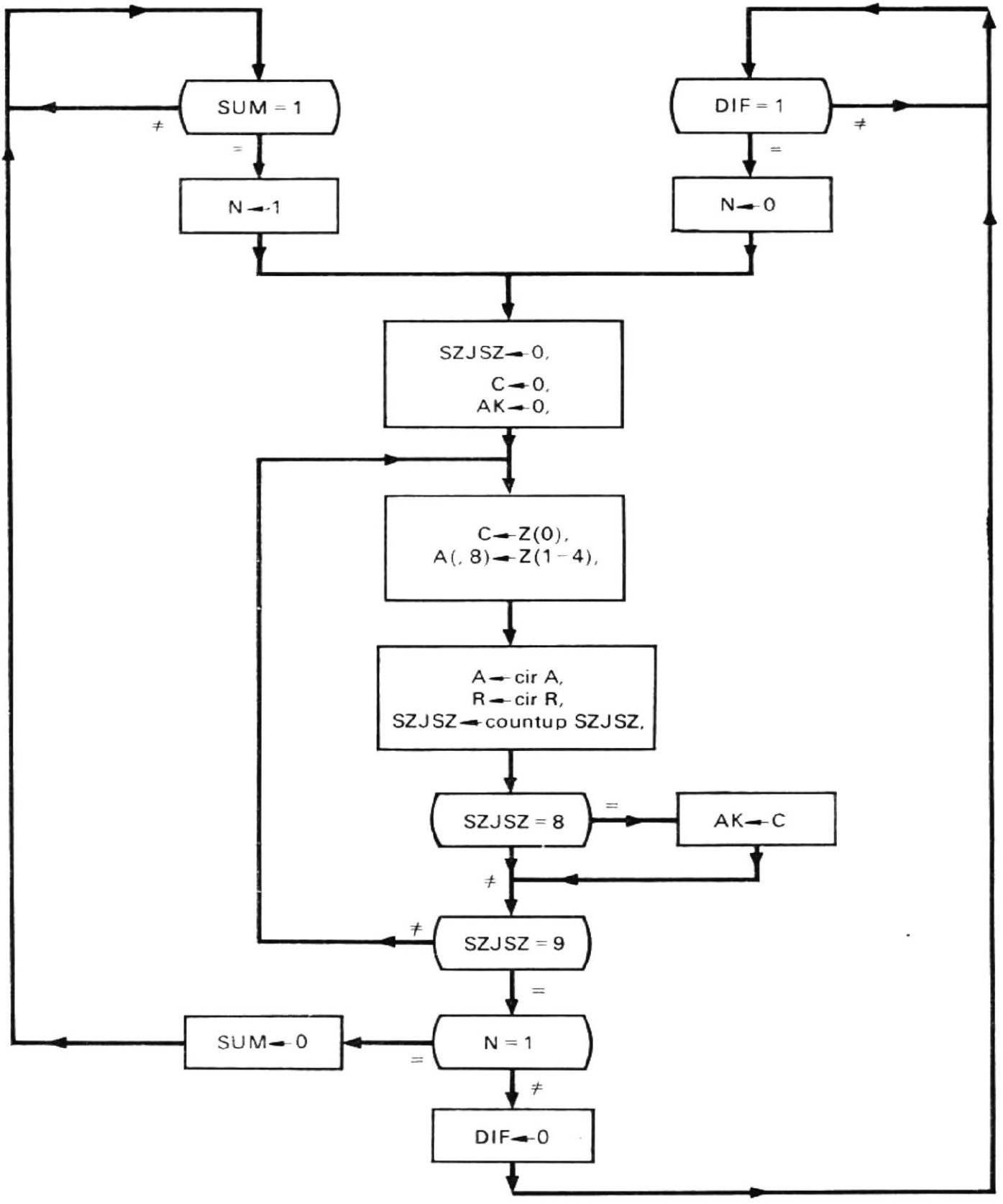
Amint a 6.18. ábrából látható, a SUM-DIF részsorozat folyamatosan vizsgálja a SUM és a DIF regisztereket a várakozási hurokban. Amikor a SUM vagy a DIF regiszter értéke 1 lesz, a részsorozat aktiválódik, miközben az N regiszter értékét 1-re, ill. 0-ra állítja be. A részsorozat ezután kezdeti értékeket kap, oly módon, hogy az SZJSZ, C és AK regiszterek 0 értéket kapnak. Most aktivizálódik a számjegyösszeadási hurok. A PDAK-ból jövő Z(0) és Z(1—4) kimeneteket beírják a C regiszterbe, ill. az A(,8) részregiszterbe. Ezután ezeket az A és az R regisztertömbök egy bitpozícióval ciklikusan jobbra léptetik és az SZJSZ számlálót 1-gyel megnövelik. Ezt követően megvizsgálják az SZJSZ számlálót a 9 értékre. Ha értéke még nem 9, a részsorozat visszatér a számjegyösszeadási hurok elejére és megismétli a hurkot. A hurok végrehajtása folyamán, amikor az SZJSZ számláló eléri a 8-at, a C regiszter tartalma beíródik az AK regiszterbe, ugyanis, ha a C 1 értéket tartalmaz, az túlcsordulást jelez. Amikor az SZJSZ számláló eléri a 9-et, befejeződik az összeadás vagy kivonás. A SUM vagy DIF regiszter 0 értéket kap, ha az N regiszter 1, ill. 0. A részsorozat most már visszatér a várakozási hurokba.

Az összeadási és kivonási sorozat a 6.17. ábrán a SUM vagy DIF regiszter értékadásával kezdődik, hogy behívják a SUM-DIF részsorozatot és azután vár a befejezésre. Amikor a részsorozat befejeződött, az összeadási és kivonási sorozat az AK regiszter



6.17. ábra
Folyamatábra a decimális összeadási és kivonási sorozathoz

megvizsgálásával folytatódik. Összeadás esetén, ha az AK regiszter 1-est tartalmaz, azt jelzi, hogy túlcsordulás lépett fel és az ATCS regiszter 1 értéket kap; a sorozat leáll. Kivonás esetén, ha az AK regiszter tartalma 1, azt jelzi, hogy átvétel lépett fel és a különbség 10-es komplemensben van ábrázolva. Ebben az esetben a különbségnek 10-es komplemensét képezik, hogy az előjelekkel és abszolút értékkel legyen ábrá-



6.18. ábra
Folyamatábra a SUM-DIF részsorozathoz

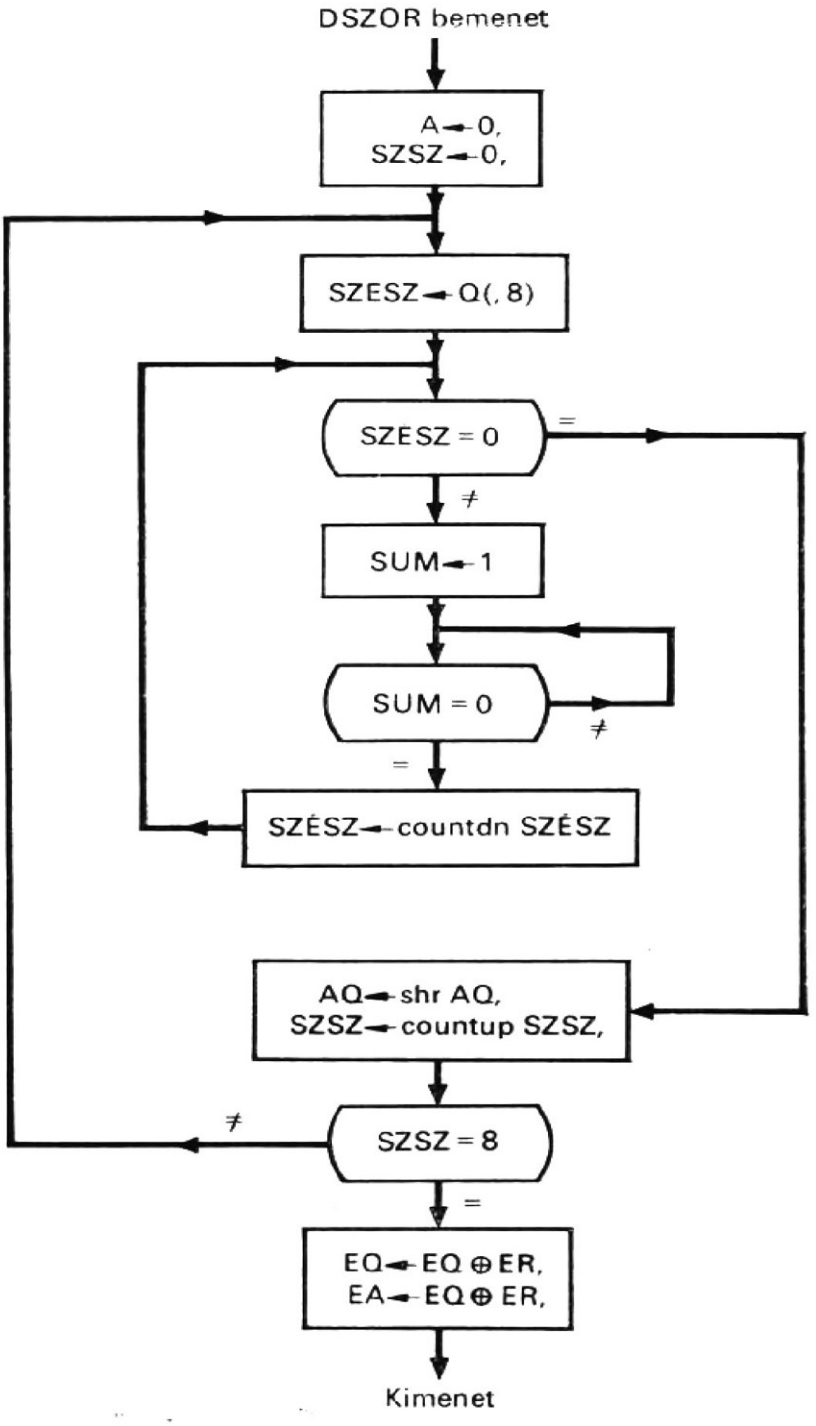
zolható. Ezt úgy végzik el, hogy invertálják az EA regiszter tartalmát és az A regiszter-tömb tartalmát beírják az R regisztertömbbe. Ezt követően az A regisztertömb 0 értéket kap és behívják kivonás céljából a SUM-DIF részsorozatot. Amikor a kivonás befejeződött, a sorozat leállt. Ha az AK regiszter értéke nem 1, a sorozat leáll.

6.6.6.
Decimális szorzás

Ebben a decimális aritmetikai egységben a szorzást a szorzandók ismételt összeadásával végzik [4]. Kezdetben a szorzót a Q regisztertömb, a szorzandót az R regisztertömb tartalmazza. A szorzás befejeztével a szorzatot a Q és A tömbregiszterek tárolják és a szorzó elvész.

Decimális szorzási sorozatot mutat a 6.19. folyamatábra. A szorzásban szükséges összeadást a SUM-DIF részsorozat hajtja végre.

Amint a 6.19. ábrából látható, két hurok van: egy belső és egy külső. Miután az A és SZSZ regiszterek 0 értéket kaptak, a külső, jobbra léptető hurokba lépünk be és így a Q(,8) részregiszter tartalmát beírjuk a SZÉSZ számlálóba. Most a SZÉSZ számláló tartalma a Q(,8) részregiszterben levő számjegyhez szükséges összeadások számát



6.19. ábra
Folyamatábra a decimális szorzási sorozathoz

jelöli ki. Ezután megindul a belső számjegyösszeadó hurok, a SZÉSZ számláló tartalmának vizsgálatával. Ha a SZÉSZ számláló nem 0, összeadják az A és az R regisztertömbök tartalmát és a SZÉSZ számláló értékeit 1-gyel csökkentik; a sorozat ezután visszatér a számjegyösszeadási hurok kezdetére. A belső hurkot most már addig ismétlik, amíg csak a SZÉSZ számláló el nem éri a 0-t. A 0 a SZÉSZ számlálóban azt jelenti, hogy nincs szükség összeadásra és a számjegyösszeadási hurok leállt. Azonban a külső hurok folytatódik az AQ regisztersorokból álló tömbnek egy bitpozícióval való jobbra léptetésével, és az SZSZ számlálónak 1 értékkel való növelésével. Ezután megvizsgálják az SZSZ számlálót a 8 értékre. Ha az nem 8, a sorozat visszatér a jobbra léptető hurok kezdetéig. A külső hurok addig folytatódik, amíg az SZSZ számláló el nem éri a 8-at. Ekkor meghatározzák az előjelet és elhelyezik azt az EA és EQ regiszterekben. A szorzás ezzel befejeződött.

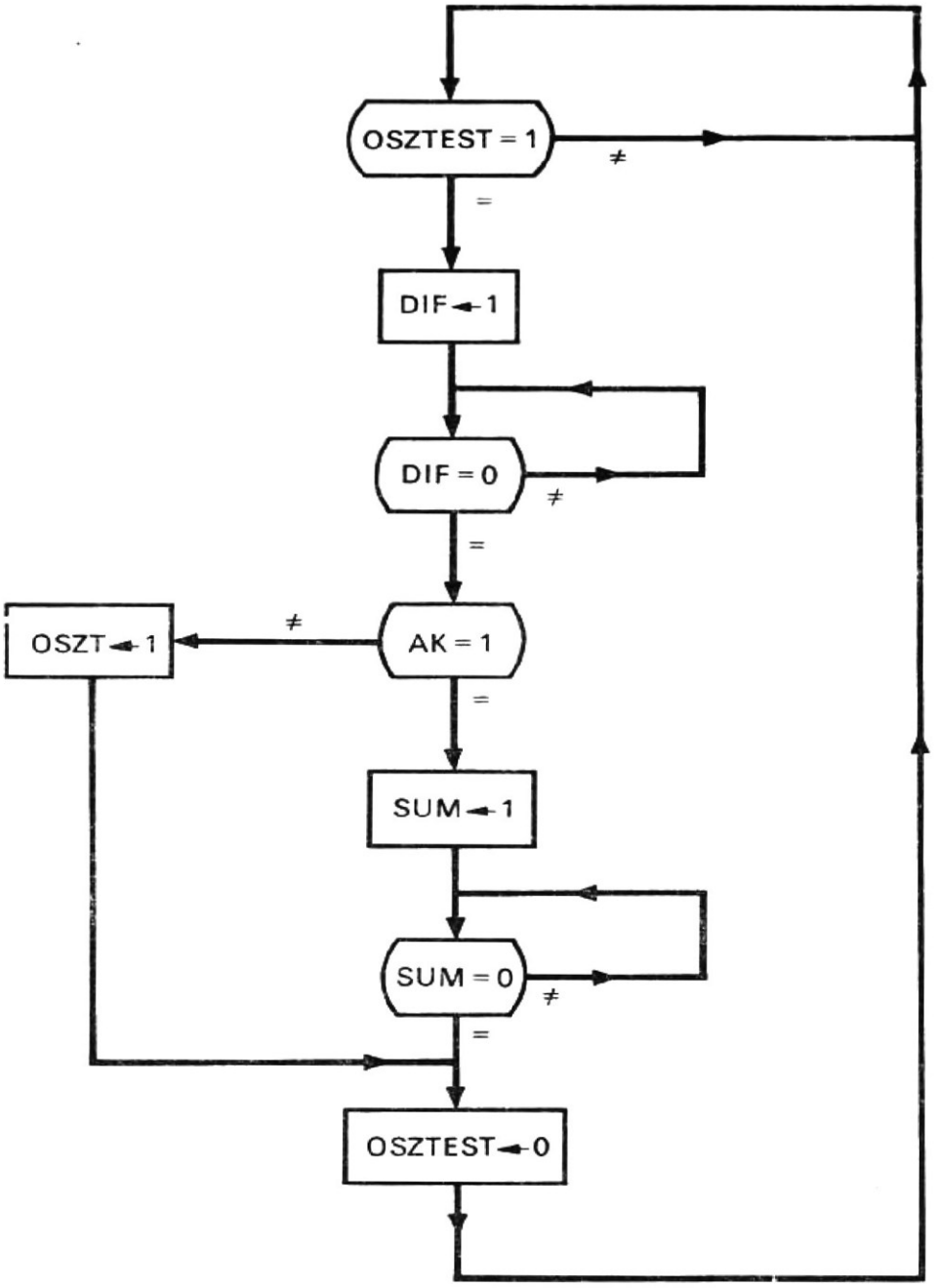
6.6.7. Decimális osztási algoritmus

A decimális osztás a visszaállítás nélküli („restoring") algoritmust használja [4]. Kezdetben az osztót az R regisztertömb tárolja, míg a 16-számjegyes osztandót az AQ regisztersorokból álló tömb. Az osztás után a hányados a Q regisztertömbben lesz, a maradék pedig az A regisztertömbben; az osztandó elvész. Az osztási túlcsordulást az OTCS regiszter jelzi.

A decimális osztási sorozatot a 6.20. és 6.21. folyamatábrák mutatják. Az osztáshoz szükséges összeadást és kivonást a SUM-DIF részsorozat hajtja végre, az osztási túlcsordulás ellenőrzését pedig az OSZTEST részsorozat végzi. Amint a 6.20. ábrából látható, az osztás azzal kezdődik, hogy az SZSZ regiszter 0 értéket kap, és behívják az OSZTEST részsorozatot és ezzel osztási túlcsordulás vizsgálat kezdődik. A 6.21. ábrán látható OSZTEST részsorozat a SUM-DIF részsorozat behívásával kezdődik, amely kivonja az R regisztertömbben levő osztót az A regisztertömbben elhelyezkedő osztandó nagyobb helyiértékű részéből, és a különbséget az A regisztertömbben tárolja. Ezután megvizsgálják az AK regisztert, hogy eldöntsék, van-e átvétel. Ha az AK regiszter tartalma 0, osztási túlcsordulás keletkezett, és az OTCS regiszter 1 értéket kap. Egyébként nem lépett fel osztási túlcsordulás, és az osztandót visszaállítják a SUM-DIF részsorozat behívásával, hogy adja hozzá az R regisztertömbben levő osztót az A regisztertömbben levő különbséghez. Ekkor az OSZTEST részsorozat végrehajtása befejeződött, és ez a részsorozat leáll, miután az OSZTEST regiszternek 0 értéket adott.

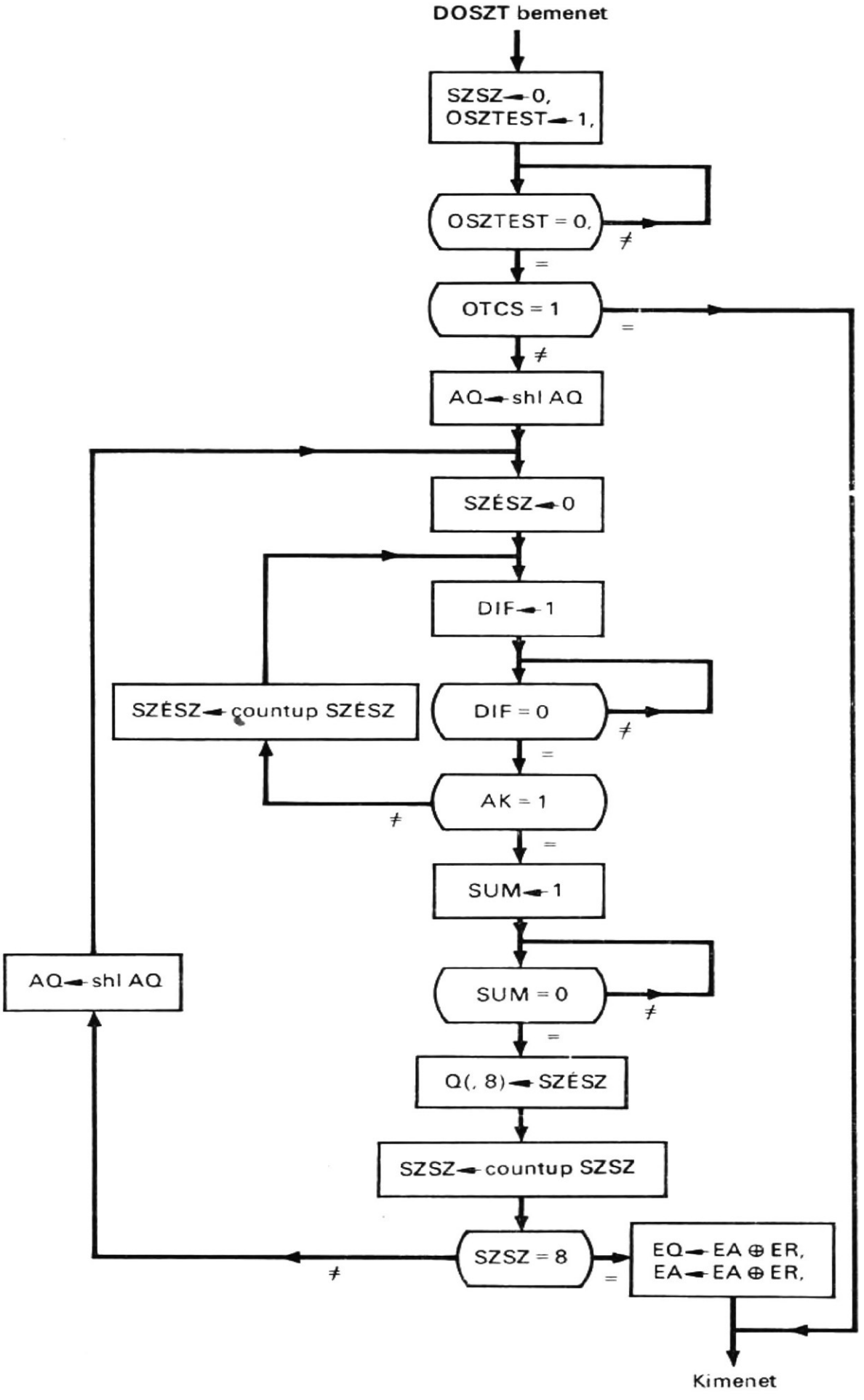
Az OSZTEST részsorozatból való visszatérés után az osztási sorozat megvizsgálja az OTCS regisztert 1-re. Ha az 1, az azt mutatja, hogy osztási túlcsordulás lépett fel és így az osztás leáll. Ha viszont értéke nem 1, az osztás folytatódik. Minthogy az osztási túlcsordulás ellenőrzése megállapította, hogy az A regisztertömb értéke kisebb, mint az R regisztertömbben levő osztó, az AQ regisztersorokból álló tömbben levő osztandót tízzel megszorozzák oly módon, hogy egy bitpozícióval balra léptetik.

Amint a 6.20. ábrából látható, két hurok van: egy belső és egy külső. A külső, balra léptető hurok azzal kezdődik, hogy a SZÉSZ számlálót visszaállítja 0-ra. Ezután aktiválódik a belső, számjegykivonó hurok. A belső hurok behívja a SUM-DIF részsorozatot kivonás céljából, és megvizsgálja, keletkezett-e átvétel a kivonás során. Ha nem fordult elő átvétel, a SZÉSZ számláló értékét 1-gyel megnövelik és a sorozat visszatér a belső hurok kezdetére. A számjegykivonási hurkot addig folytatják, amíg átvétel elő nem fordul, és a belső hurok leáll. Az átvétel jelenléte azt jelzi, hogy a SZÉSZ



6.20. ábra
Folyamatábra a decimális osztási sorozathoz

számláló tartalma a hányadosszámjegy értéke; ezt az értéket beírják a Q(,8) részregiszterbe. Ezután az SZSZ számlálót 1-gyel megnövelik, majd megvizsgálják, hogy értéke 8-e? Ha értéke nem 8, az egyesített regisztert egy számjegypozícióval balra léptetik és a sorozat visszatér a külső hurok kezdetéhez. A külső hurkot addig ismétlik, amíg az SZSZ értéke el nem éri a 8-at. Ekkor meghatározzák a hányados és a maradék előjelét és elhelyezik azokat az EQ, ill. EA regiszterekbe. Ezzel az osztás befejeződött.



6.21. ábra
Folyamatábra az osztás-stop ellenőrzési részsorozathoz

6.7.
Decimális szorzási és osztási eljárások

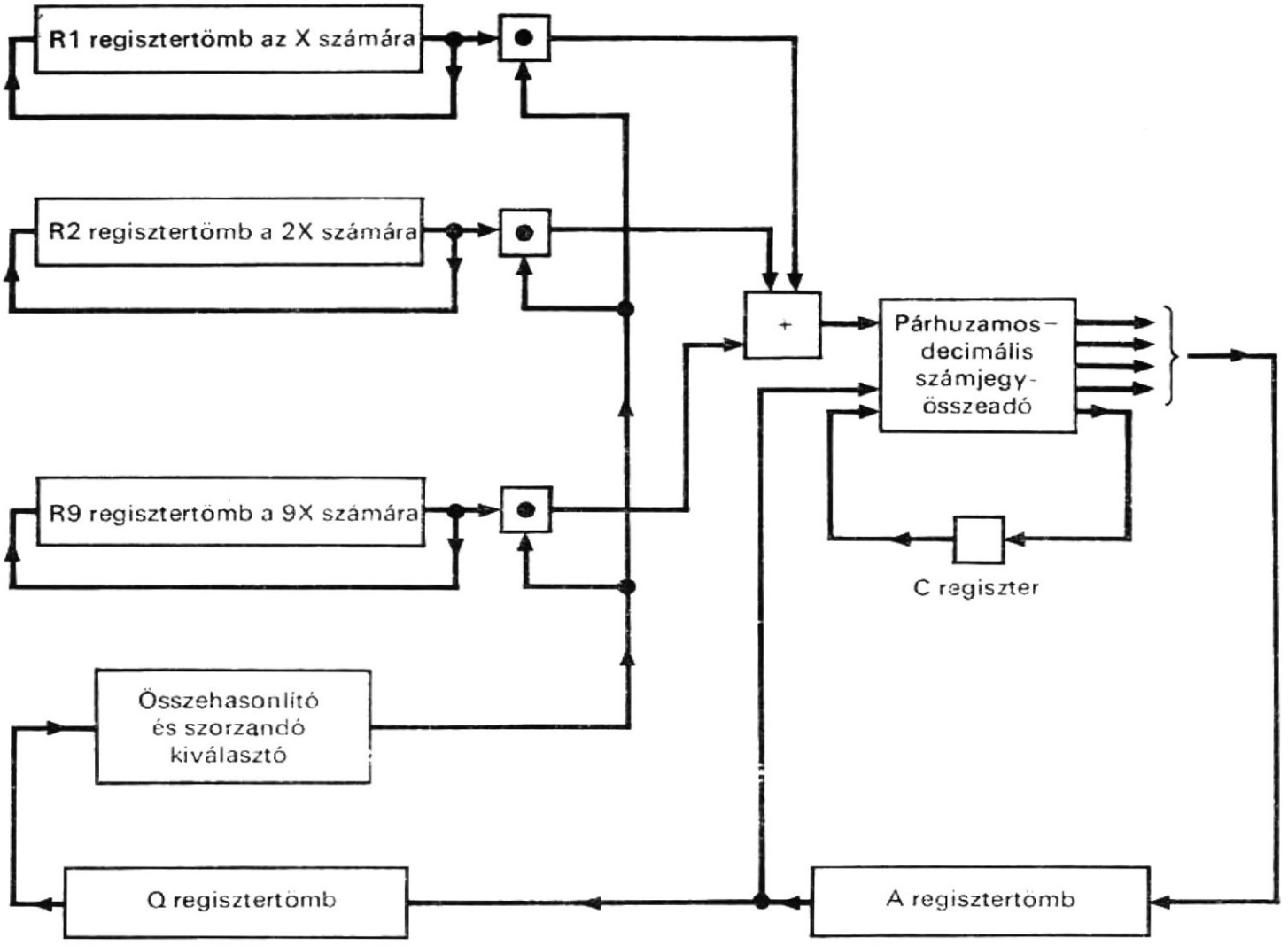
A decimális szorzásnál alkalmazott ismételt összeadási algoritmus, és a decimális osztásnál alkalmazott ismételt kivonási algoritmus mellett, ismert egy sor más eljárás is a decimális szorzás és osztás realizálására. A következőkben bemutatunk három algoritmust a decimális szorzáshoz és egy algoritmust a decimális osztáshoz. A következőkben az összeadási vagy kivonási idő alatt azt az időt értjük, amely egy decimális számnak egy másik decimális számhoz való hozzáadásához, ill. egy decimális számnak egy másik decimális számból való kivonásához szükséges, soros számjegy működésmódban.

6.7.1.
A szorzandó 1...9-szeresét használó szorzási eljárás

Egy ilyen soros számjegyű szorzóberendezés struktúrája látható a 6.22. ábra blokkdiagramján [3, 4]. A szorzó struktúrája és sorrendi működése hasonló a 6.15. ábrán bemutatott egységhez. Két lényeges eltérésük van: az egyik a kilenc regisztertömb alkalmazása az X szorzandó többszöröseinek tárolására, és a másik, az összehasonlító- és szorzandó-kiválasztó (komparátorszelektor) egység alkalmazása, amely a SZÉSZ számlálót helyettesíti. Az összehasonlító- és szorzandó-kiválasztó „érzékeli" a legközelebbi szorzószámjegyet és azután kiválasztja a szorzandónak az így meghatározott többszörösét, amelyet aztán hozzá kell adni a részszorzathoz.

A szorzandó többszöröseit a következőképpen lehet előállítani. Kiindulásként törlik az A regisztertömböt és az X szorzandót az R1 regiszter tárolja. Összeadást hajtanak végre; az összeg, az X, az A-ban marad. Ezt, az A-ban levő összeget beírják az R2 regiszterbe; ugyanakkor az A-ban levő összeget és az R1-ben levő, X jelű szorzandót ismét összeadják, így a 2X-et hozzák létre az A regisztertömbben. Ez a folyamat addig folytatódik, amíg mind a kilenc többszöröst elő nem állították és ezeket az R1, R2,...,R9 -ben tárolják. A szorzási idő annyi összeadási időt igényel, ahány számjegy van a decimális számban, hozzávéve azt a kilenc összeadási időt, amely a többszörösöket generálja.

Ennek a szorzási eljárásnak az a hátránya, hogy kilenc regisztertömbre van szükség a kilenc többszörös tárolására. A regisztertömbök számát azonban csökkenteni lehet, ha észrevesszük, hogy a szorzandó 6...9-szeresének előállítása helyett a szorzandó 10-szereséből kivonjuk a szorzandó 4...1-szeresét. Így csak a szorzandó 1...5-szörösét kell előzetesen előállítani, de összeadó helyett összeadó—kivonó egységet kell alkalmazni.

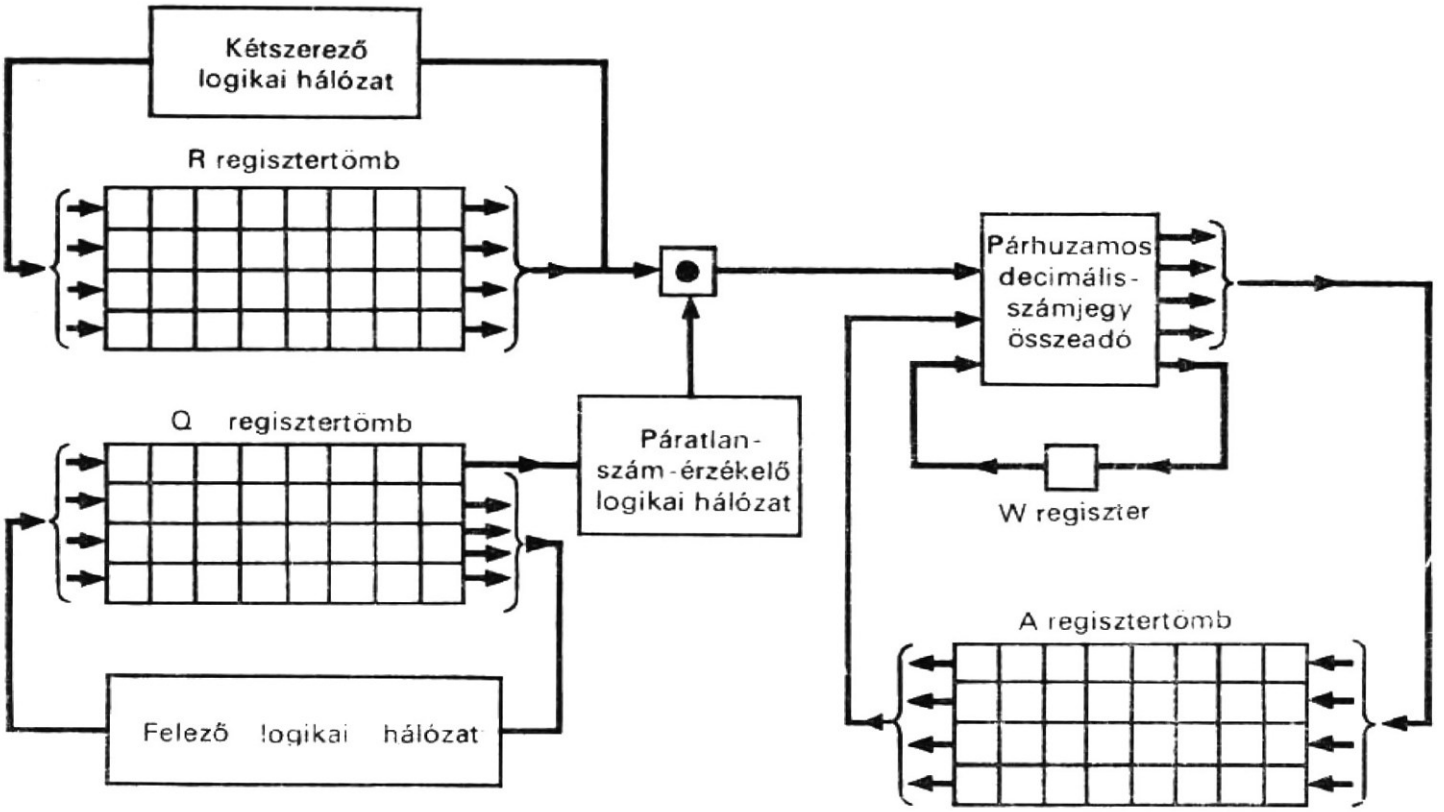


6.22. ábra
A szorzandó 1—9-szeresét használó decimális szorzóberendezés struktúrája

6.7.2.
A kétszerező—felező módszert alkalmazó szorzóberendezés

A 6.23. ábra olyan soros számjegyű szorzóberendezés struktúráját mutatja, amely a kétszerező—felező módszert alkalmazza [3, 4]. Ehhez az algoritmushoz az szükséges, hogy a szorzandót megkétszerezzék és a szorzót megfelezzék. Amint a 6.23. ábrából látható, az R, Q és az A regisztert használjuk a szorzandó, a szorzó, ill. a szorzat tárolására. Kétszerező logikai hálózatot iktatnak be a szorzandó ciklikus léptetési útvonalába, az R regisztertömbbe, és egy felező logikai hálózatot a szorzó ciklikus léptetési útvonalába, a Q regisztertömbbe.

A szorzás annak megvizsgálásával kezdődik, hogy a szorzó páratlan vagy páros-e. Ha páros, a szorzandót megkétszerezik oly módon, hogy ciklikusan léptetik a kétszerező logikai hálózaton át, a szorzót pedig megfelezik, a felező logikai hálózaton történő ciklikus léptetéssel. Ha a szorzó páratlan, a szorzandót ismét megkétszerezik, azonban a szorzóból először 1-et kivonnak a számjegy legkisebb helyiértékű pozíciójából, és azután megfelezik. Ez egy összeadási idő alatt játszódik le. Az összeadás folyamán, míg a szorzandót ciklikusan léptetik, hozzá is adják közben az A regisztertömbben levő részszorzathoz, ha a szorzó páratlan. Ezt a folyamatot: a szorzandó meg-



6.23. ábra
A kétszerező—felező módszert alkalmazó decimális szorzóberendezés struktúrája

kétszerezését, a szorzó megfelezését és a részszorzat előállítását egyetlen összeadási idő alatt, a szorzó minden megfelezése után megismétlik, amíg a Q regisztertömb tartalma nulla nem lesz. Ekkor a szorzat az A regisztertömbben van. Ha a szorzat 2n számjegyét meg akarjuk őrizni, további regisztertömbök szükségesek, és az így szükséges regiszterek száma nagyobb lesz, mint a 6.15. ábrán levő struktúrában.

6.7.3. Beépített szorzótáblát alkalmazó szorzóberendezés

Beépített szorzótáblát alkalmazó, soros számjegy típusú szorzóberendezés struktúrája látható a 6.24. ábrán. A szorzótáblát *egy decimális számjegy szorzónak* nevezett logikai hálózattal valósítják meg, amely összeszoroz két decimális számjegyet és képezi két decimális számjegy szorzatát.

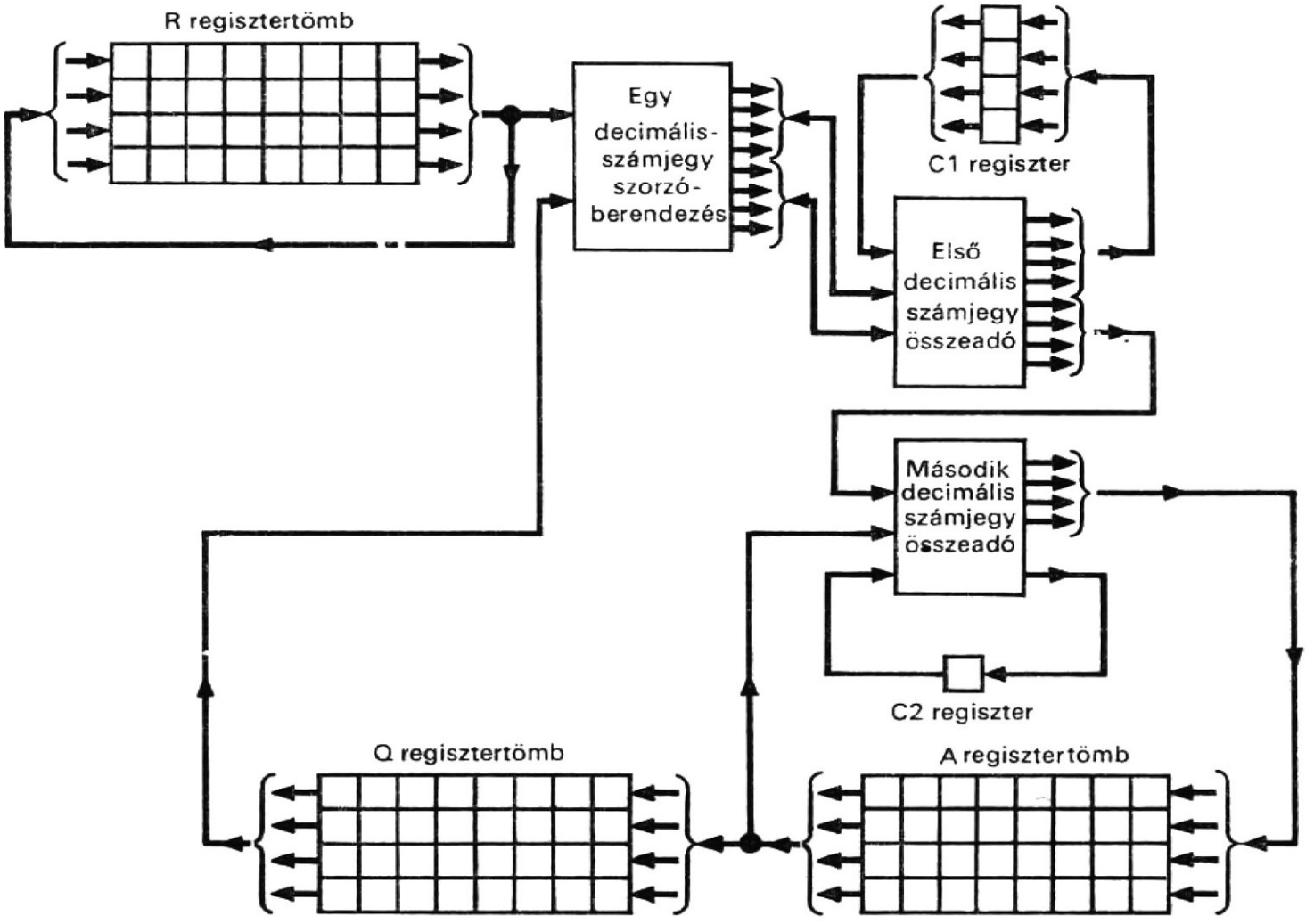
Amint a 6.24. ábrából látható, az R, Q és A regisztertömbökön kívül még a C1 és C2 átvitelregiszterekre is szükség van. A szorzandót és a szorzót az R, ill. a Q regisztertömb tárolja. A C1 regiszter 4-bites, a C2 regiszter egybites; valamint összeadót alkalmazunk: az első decimális számjegy összeadót és a második decimális számjegy összeadót.

A szorzás az R és a Q regisztertömbökben levő, két legkisebb helyiértékű számjegy szorzásával kezdődik, amit az egy decimális számjegy szorzóegység végez. Az egy decimális számjegy szorzóból jövő szám két számjegyét ezután hozzáadják a C1 regiszterhez, amely az előző számjegyátvitelt tartalmazza. Ez a számjegyátvitel kezdetben 0, egyébként az értéke 0, 1,...,9 lehet. Ezt az összeadást az első decimális számjegy összeadó végzi, amely két számjegyet hoz létre; a kisebb helyiértékű számjegyet azután a C1 regiszter tárolja, míg a nagyobb helyiértékű számjegyet a második deci-

mális számjegy összeadóval hozzáadják az A regisztertömbben levő részszorzathoz és a C2 regiszterben levő átvitelhez. A második decimális számjegy összeadó hozza létre az egybites átvitelt és a számjegyösszeget. Az átvitelbitet ezután a C2 regiszter tárolja és az összegszámjegyet az A regisztertömbbe léptetik át, mint a részszorzatnak a részét.

Az R és a Q regisztertömbökben levő legkisebb két számjegy összeszorzása után eggyel léptetik az R és Q regisztertömböt, hogy a következő, alacsonyabb helyiértékű számjegyeket össze tudja szorozni az egy decimális számjegy szorzó. Ez a szorzó egy másik, kétszámjegyes számot ad, amelyet ismét hozzáad az első számjegy összeadó a C1 regiszterben levő átvitelhez, hogy egy másik kétszámjegy összeget hozzon létre. Ezt a kétszámjegy összeget azután a második decimális számjegy összeadó hozzáadja az A regisztertömbben levő részszorzathoz, és a C2 regiszterben levő átvitelhez. Ezt követően a második decimális számjegy összeadóról a számjegyösszeget átléptetik az A regisztertömbbe, míg az R és Q regisztertömböket úgy léptetik, hogy a következő két legkisebb helyiértékű számjegyet lehessen összeszorozni. Ez a folyamat addig folytatódik, amíg az R és Q regisztertömbökben levő decimális számok valamennyi számjegye nincs összeszorozva. Ekkor a szorzat a Q és az A regisztertömbökben van.

A fenti decimális szorzóban eléggé bonyolult az első decimális számjegy összeadó, mivel három számjegyet kell összeadnia. A második decimális számjegy összeadó az előzőekben leírt párhuzamos decimális számjegy összeadót tudja használni, ha a decimális szám kódolása 8–4–2–1. A szorzási idő annyi összeadási időt igényel, amennyi a számjegyek száma a decimális számban.



6.24. ábra
A beépített szorzótáblával rendelkező decimális szorzóberendezés struktúrája

6.7.4.
Az osztó 9-szeresét felhasználó osztóberendezés

Az osztó 9-szeresét felhasználó, soros számjegy típusú decimális osztóberendezés struktúrája hasonló ahhoz a decimális szorzóberendezéshez, amely a szorzandó 1...9-szeresét használja, és amit a 6.22. ábrán láthattunk. Kilenc regisztertömb rendre: az R1,...,R9, ezek tárolják a kilenc többszöröst; valamint a Q és az A regisztertömbök. A szorzót a Q regisztertömb tárolja. Szükség van azonban egy decimális számjegy összeadó—kivonóra; az összeadót a kilenc többszörös előállítására használják, a kivonót pedig az osztás végrehajtására, ismételt kivonásokkal. Továbbá szükség van egy összehasonlítóra (komparátor), ez végzi a kilenc többszörös szimultán összehasonlítását a részmaradékokkal, és azután kiválasztja a legnagyobb többszöröst, amely a következő részmaradékot pozitívvá teszi. A kiválasztott többszörösből hányadosszámjegy generálódik és a kiválasztott többszöröst kivonják az aktuális részmaradékból, hogy megkapják a következő részmaradékot. Az osztás után a hányadost az A regisztertömb tárolja.

A fent említett decimális osztóberendezésben levő összehasonlító eléggé bonyolult és a jelenlegi technológiával nem valósítható meg. Az osztási idő annyi kivonási időt igényel, amennyi a decimális számban levő számjegyek száma, továbbá a kilenc többszörös előállításához, valamint az összehasonlításhoz és kiválogatáshoz szükséges időket is figyelembe kell venni.

Feladatok

6.1. A kétbites bináris összeadó egyszerre adja össze a kétbites összeadandókat és az egyetlen átvitelbitet (vagyis nem használ két, egybites teljes összeadót), a kétbites összeg és az egybites átvitel létrehozására. Írjuk le terminálutasítással a kétbites összeadót!

6.2. A kétbites kivonó egyszerre vonja ki a kétbites kivonandót és az egybites átvételt a kétbites kisebbítendőből, és egy kétbites különbséget és egybites átvételt hoz létre. Írjuk le terminálutasítással a kétbites kivonót!

6.3. A kétbites összeadó—kivonó kétbites összeadóként működik, ha az egybites N vezérlőregiszter tartalma 1, és kétbites kivonóként, ha az N regiszter tartalma 0. Írjuk le terminálutasítással a kétbites összeadó—kivonót!

6.4. Módosítsuk a 6.5. folyamatábrán levő kivonási sorozat, és a 6.6. folyamatábrán levő SUM-DIF részsorozatot arra az esetre, ha a 6.3. feladatban leírt kétbites összeadó—kivonót használjuk az egybites összeadó—kivonó helyett!

6.5. Módosítsuk a 6.8. folyamatábrát a szorzási sorozathoz arra az esetre, ha kétbites összeadó—kivonót használunk!

6.6. Módosítsuk a 6.10. folyamatábrát a bináris szorzási sorozathoz, és a 6.11. folyamatábrát az osztás-stop részsorozathoz arra az esetre, ha kétbites összeadó—kivonót használunk!

6.7. Módosítsuk a bináris, soros aritmetikai egységnek a 6.2. és 6.12. ábrán levő struktúráit, és az összeadási és kivonási sorozatnak a 6.5. ábrán látható folyamatábráit, a SUM-DIF részsorozatot a 6.6. ábrából, a szorzási sorozatot a 6.8. ábrából, az osztási sorozatot a 6.10. ábrából és az osztás-stop ellenőrző részsorozatot a 6.11. ábrából, ha a bináris számok ábrázolása előjeles 1-es komplemens!

6.8. Ismételjük meg a 6.7. feladatot, ha a bináris számok előjellel és abszolút értékkel vannak ábrázolva!

6.9. Tervezzük meg a (6.22) utasításokban leírt decimális, soros számjegy típusú aritmetikai egységhez a struktúra vezérlőrészét és írjuk le a következő sorozatokat CDL utasításokkal:
a) a 6.17. ábrán levő decimális összeadási és kivonási sorozatot;
b) a 6.18. ábrán levő SUM-DIF részsorozatot;
c) a 6.9. ábrán levő decimális szorzási sorozatot;
d) a 6.20. ábrán levő decimális osztási sorozatot;
e) a 6.21. ábrán levő osztás-stop részsorozatot!

6.10. Definiáljunk terminálutasítással, 3-többletű BCD kódú, párhuzamos, decimális összeadót!

6.11. Definiáljunk terminálutasítással, 3-többletű BCD kódú, párhuzamos, decimális kivonót!

6.12. Definiáljunk terminálutasítással, 3-többletű BCD kódú, párhuzamos, decimális összeadó—kivonót! Az N regiszter az összeadás—kivonás vezérlőregisztere. Ha az N regiszter tartalma 1, az összeadó—kivonó összeadóként működik; egyébként pedig kivonóként dolgozik.

6.13. Módosítsuk a decimális összeadási és kivonási sorozatnak a 6.17. ábrán látható folyamatábráját és a SUM-DIF részsorozatnak a 6.18. ábrán látható folyamatábráját arra az esetre, ha a 6.11. feladatban leírt, 3-többletű kódot alkalmazó decimális összeadó—kivonó egységet alkalmazzuk a 8–4–2–1 kódot használó decimális összeadó—kivonó helyett!

6.14. Módosítsuk a decimális szorzási sorozat 6.19. folyamatábráját arra az esetre, ha a 3-többletű kódot használó decimális összeadó—kivonót alkalmazzuk!

6.15. Módosítsuk a decimális osztási sorozat 6.20. folyamatábráját és az osztás-stop ellenőrző részsorozat 6.21. folyamatábráját arra az esetre, ha 3-többletű kódot használó decimális összeadó—kivonót alkalmazunk!

6.16. Tervezzünk a vezérlőrészt is magában foglaló struktúrát decimális soros típusú aritmetikai egységhez és írjuk le utasításokkal a következő sorozatokat:
a) a 6.13. feladatból kapott decimális összeadási és kivonási sorozatot;
b) a 6.13. feladatból kapott SUM-DIF részsorozatot;
c) a 6.14. feladatból kapott decimális szorzási sorozatot;
d) a 6.15. feladatból kapott osztás-stop ellenőrzési részsorozatot;
e) a 6.15. feladatból kapott decimális osztási sorozatot!

6.17. Írjuk le terminálutasítással a soros decimális típusú összeadót (SDA), amely 8–4–2–1 kódot használ!

6.18. Írjunk le egy olyan soros decimális kivonót (SDK) terminálutasítással, amely 8–4–2–1 kódot használ!

6.19. Írjunk le terminálutasítással egy olyan soros decimális összeadó—kivonót (SDAK), amely 8–4–2–1 kódot használ! Az N regiszter az összeadás/kivonás vezérlőregisztere. Ha az N regiszter tartalma 1, az összeadó—kivonó összeadóként működik. Egyébként pedig kivonóként dolgozik.

6.20. Módosítsuk a következő folyamatábrákat, a 6.19. feladatból kapott, SDAK-t alkalmazó, decimális soros aritmetikai egységhez:
a) a 6.17. folyamatábrát decimális összeadáshoz és kivonáshoz;
b) a 6.18. folyamatábrát a SUM-DIF részsorozathoz;
c) a 6.19. folyamatábrát a decimális szorzási sorozathoz;
d) a 6.20. folyamatábrát a decimális osztási sorozathoz;
e) a 6.21. folyamatábrát az osztás-stop ellenőrzési részsorozathoz!

Irodalom

[1] BURKS, A.W.—GOLDSTINE, H. H.—von NEUMANN, J.: *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument.* Inst. Advanced Study Rept. 1, pt. 1, June 28, 1946.

[2] SHAW, R. E.: *Aritmetic Operations in a Binary Computer.* Rev. Sci. Instr., August, 1950, pp. 687...793.

[3] Staff of Harvard Computation Laboratory, Synthesis of Electronic Computing and Control Circuits, Cambridge, Mass.: Harvard University Press, 1951.

[4] CHU, Y.; *Digital Computer Design Fundamentals.* New York, McGraw-Hill Book Company, 1962.

7.
Társzervezés

A társzervezés a digitális számítógépek tárainak funkcionális és működési jellemzőivel foglalkozik. Ebben a fejezetben csak azokat a társzervezési elveket mutatjuk be, amelyek a központi feldolgozóegységhez (Central Process Unit: CPU) tartoznak, tehát az ún. főtárak szervezésével foglalkoznak. A közvetlen elérésű táron és az asszociatív táron kívül e fejezet bemutatja a tár címzését, betöltését, a verem- (stack-) tárakat, a pufferszervezésű tárat és a virtuális tárat. A mélyebb betekintés érdekében ezekben a leírásokban több működési részletet is megadunk. Amint a leírásokból is látható, a főtár szervezése egyre növekvő jelentőségű problémakör, megfelelő szervezéssel a hardware kihasználása hatékonyabbá válhat.

7.1.
Közvetlen elérésű tár

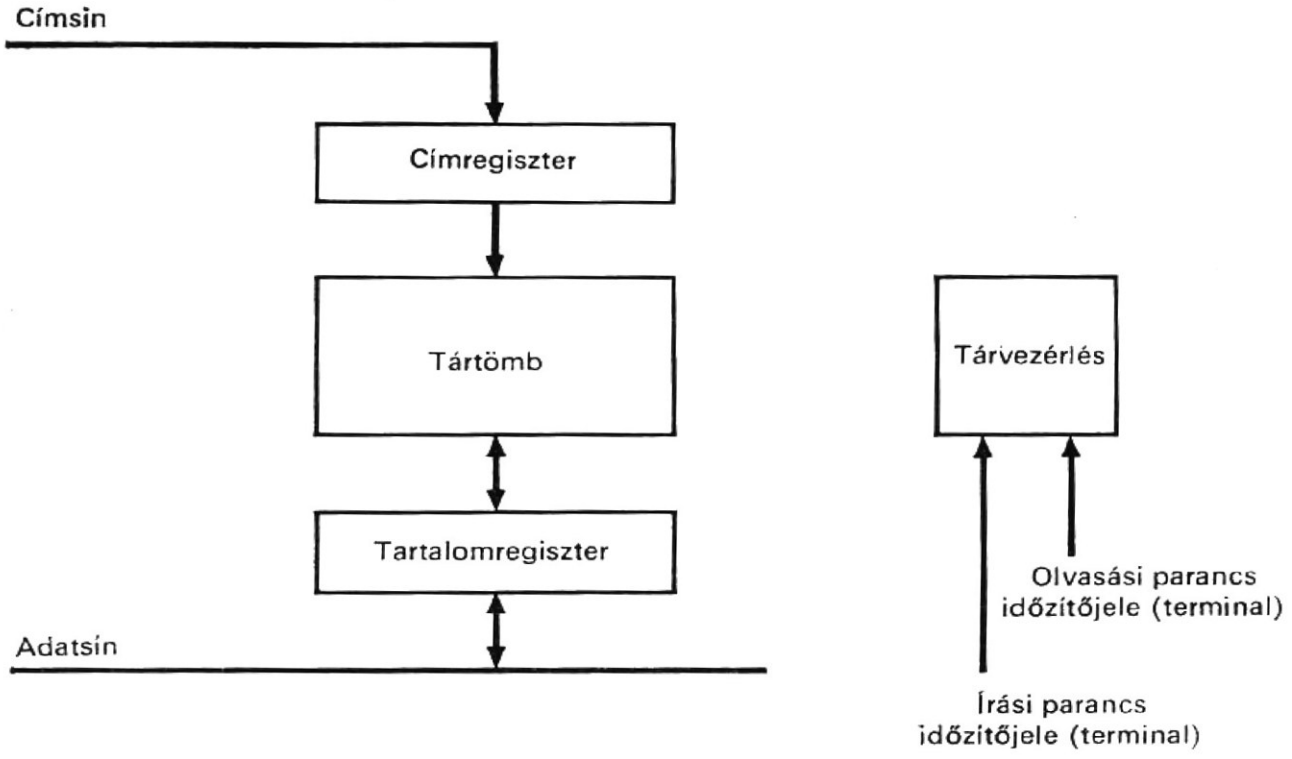
Ebben a fejezetben közvetlen elérésű tárakkal foglalkozunk. A korszerű, tárolt programú számítógépek közvetlen elérésű tárakat igényelnek, amelyekben a programokat és az adatokat tárolják, és ezeket a processzornak véletlen módon, közvetlenül kell elérnie a program végrehajtásakor. Ez a fejezet a közvetlen elérésű tár funkcionális szervezését részletesen tárgyalja. Bemutatjuk a tömbszervezést, a modulszervezést és a többszörös hozzáférés szervezését, valamint áttekintjük a közvetlen elérésű tárak típusait.

7.1.1.
Tömbszervezés

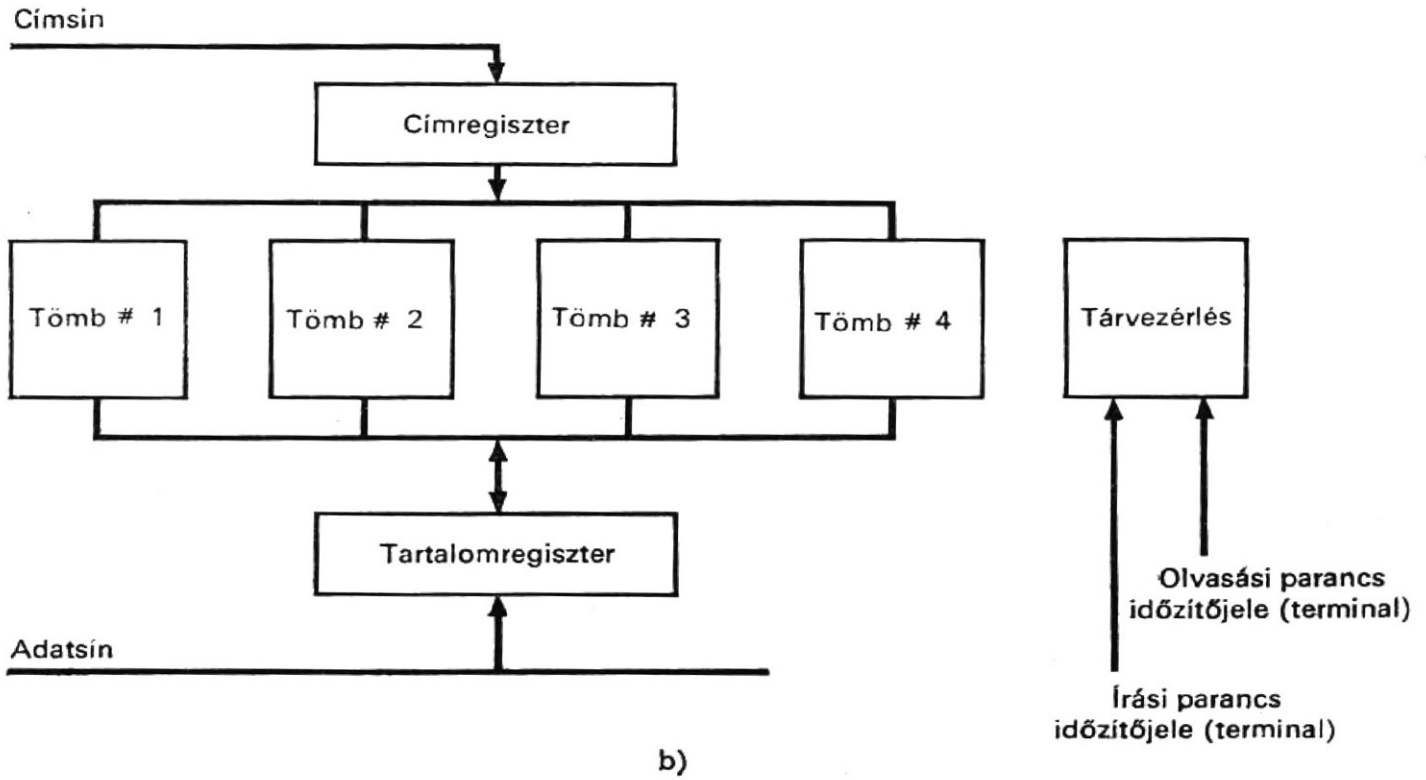
A mai közvetlen elérésű tárak többsége mágnesmagos tár. Gazdasági okok miatt ezek rendszerint koincidenciás áramú tárak. A koincidenciás áramú tárakban egy mag egy bit információt tárol. A magokat lapokba szerelik össze, amiket *társíknak* neveznek, ezeket pedig kötegekbe rakják össze, amiket *tármátrixnak* neveznek. Egy ilyen felépítésű tárban rendszerint annyi mag van egy síkban, amennyi a tárban levő szavak

száma és annyi sík egy kötegben, amennyi a bitek száma a szóban. Például egy 72-bites szavakból álló, 16 384 szavas, mátrixszervezésű mágnesmagos tárnak 72 tárolósíkja lehet, és minden síkban 128×128 mag. A koincidenciás áramú tár síkját olyan nagyra kell választani, amilyenre csak lehet, mivel a nagyobb síknak jobb a gazdasági kihasználtsága. A 128×128 magos társík az egyik leggazdaságosabb elrendezés.

A 7.1. ábrán levő blokkdiagramok a közvetlen hozzáférésű tár felépítését mutatják. A tárak a tármátrixot, a címregisztert, a tartalomregisztert (adatregiszternek is nevezik), az olvasási vagy írási műveletek időzítésére szolgáló bemeneteket (terminálo-



a)



b)

7.1. ábra
A tár tömbszervezése
a) **egyszeres tömbszervezés;** *b)* **többszörös tömbszervezés**

kat) és a tárvezérlést tartalmazzák. Gyakran adatsíneket és címsíneket is kialakítanak, ezeket is láthatjuk a 7.1. ábrán. A tármátrix állhat egyetlen mátrixból, amint a 7.1*a* ábrán látható, vagy több mátrixból, amit a 7.1*b* ábra mutat; mindkét esetben azonban csak egyetlen egy címregiszter és egy tartalomregiszter van.

A tárba egy tárciklus alatt lehet beírni vagy kiolvasni. Az olvasási műveletet végrehajtó tárciklusban a címet először beírják a tár címregiszterébe; a megcímzett szót azután a tármátrixból kiolvassák a tartalomregiszterbe; és végül a szót visszaírják a tárba ugyanarra a helyre (azaz visszaállítják) az olvasási művelet törlő (destruktív) volta miatt. Az írási műveletet végrehajtó tárciklusban a címet átvivő művelet ugyanaz, mint az előzőben, a megcímzett szónak a tartalomregiszterbe való beolvasását meggátolják; végül az éppen kiolvasott szó visszaírása helyett a most, a tartalomregiszterben levő szót írják be a tárba.

Egyes tárak független olvasási és írási ciklusokkal működnek. Ezek vagy olvasási vagy írási ciklust tudnak végrehajtani. Az olvasási ciklusban a szót kiolvassák a tárból a tartalomregiszterbe és a tárhelyet nullákkal töltik ki. Az írási ciklusban a tartalomregiszterben levő adatokat beírják arra a tárhelyre, amelyet nullákkal töltöttek ki a megelőző olvasási ciklusban. Az olvasási művelet végén a tár olvasási állapotban marad, amíg csak egy újabb írási művelet nem kezdődik. Az írási művelet végén a tár olvasásra áll át, tehát az írási ciklust mindig olvasási ciklus előzi meg. Az olyan tárat, amely független olvasási és írási ciklusokkal működik, *osztott ciklusú műveletre alkalmas* tárnak nevezik. Az osztott ciklusú műveletre alkalmas tár — az olvasás—írási ciklushoz képest pótlólag — az olvasási ciklust, az olvasás—várakozás ciklust, az olvasás—várakozás—írás ciklust, az olvasás—írás—várakozás ciklust, valamint az olvasás—várakozás—írás—várakozás ciklust is végre tudja hajtani. Ezáltal a tár kihasználtsága hatékonyabbá válhat.

A tár működési sebességét a tárciklusidő és a tárelérési idő határozza meg. A *tárciklusidő* az olvasási—írási idő összege, míg a *tárelérési idő* a tárciklusidőnek az a része, amely alatt az olvasási művelet végrehajtódik. Az elérési idő leggyakrabban a tárciklusidő 0,3...0,7-szerese. Osztott ciklusú műveletre alkalmas tárnál két ciklusidő van; az olvasási ciklusidő és az írási ciklusidő. Az írási művelethez szükséges idő az olvasási és az írási ciklusidő összege.

Amint az előzőkben említettük, a mátrixban a társíkok számát a szóhossz határozza meg. A szóhossz a tárnak egy olyan jellemzője is, amely *az adatátviteli út szélességét* — az egy tárciklusidő alatt párhuzamosan átvitt bitek számát — mutatja. A tár maximális átviteli sebessége az adatátviteli út szélességének viszonya a tárciklusidőhöz. Például egy 1 μs ciklusidejű és 64 bit szóhosszúságú tár maximális átviteli sebessége 64 millió bit másodpercenként vagy 8 millió byte másodpercenként (egy byte nyolc bitből áll)[9].

Összegezve, a közvetlen elérésű tár jellemzői: a kapacitás, a tárciklusidő, a tárelérési idő, a tár szóhossza, adatátviteli sebessége, az egy bitre eső költség, valamint olyan speciális jellemzők, mint az osztott ciklusú művelet lehetősége.

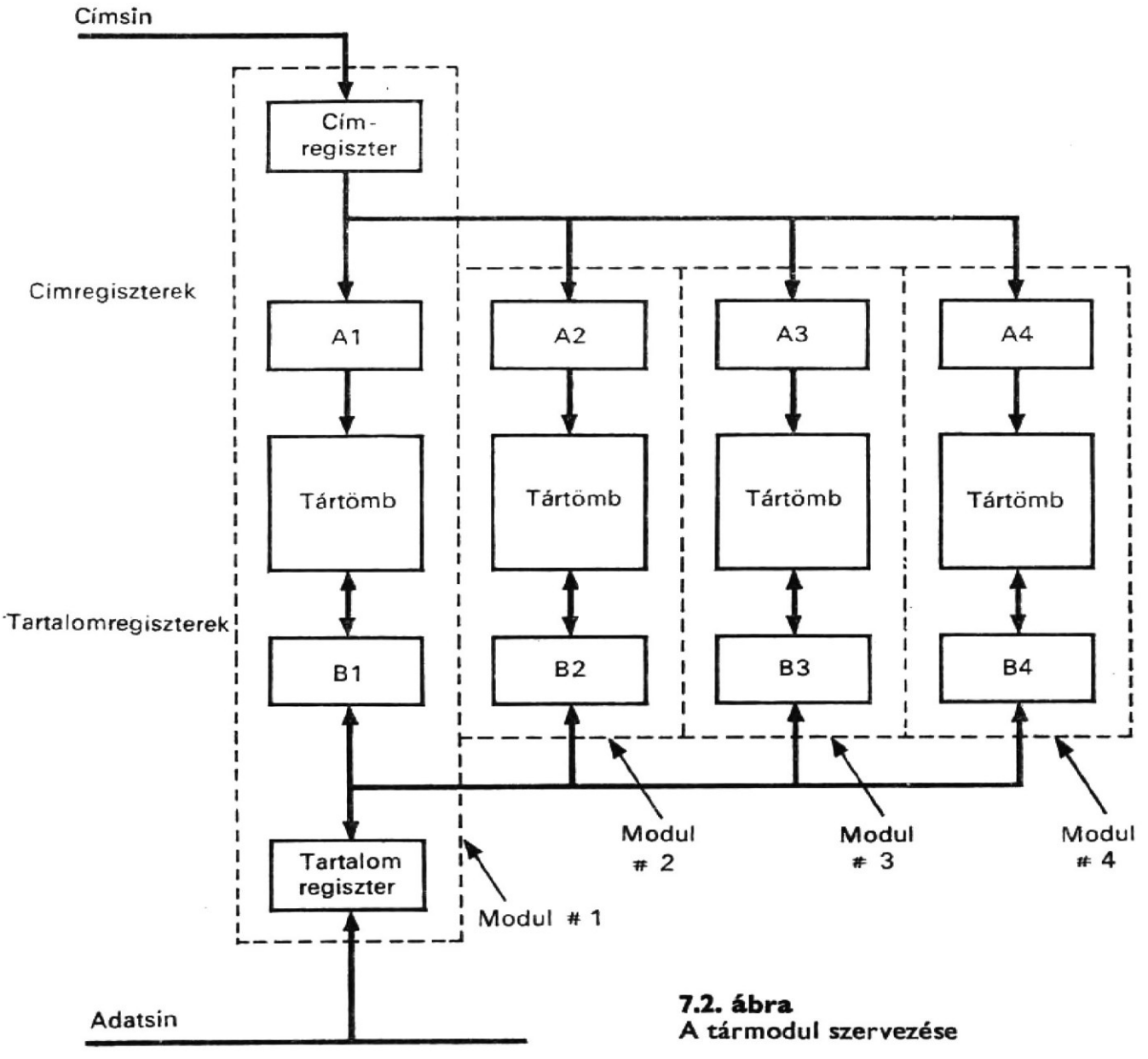
7.1.2. Modulszervezés

A koincidenciás áramú mágnesmagos tár kapacitása több mátrix alkalmazásával megnövelhető. Azonban csupán a mátrixok számának növelésével nem nő meg az adatátviteli sebesség. A modulszervezés mind a tárkapacitást, mind az adatátviteli sebességet képes megnövelni.

[9] A byte csak valamilyen kötött bitszámot jelöl, de nemcsak 8-bites byte lehetséges (lehet pl. 6-bites). *(Fordító)*

A 7.1*b* ábrán látható, több mátrixból álló szervezésben, ha mindegyik mátrixhoz egy cím- és tartalomregiszter-párt adunk hozzá, akkor az egyes tárolómátrix-csoportokban a címregiszter és a tartalomregiszter egy modult alkot. Egy-egy modul rendszerint önálló. A 7.2. ábra négy modulból álló társzervezést mutat. Valamely digitális számítógép főtárolóegysége egy vagy több modult tartalmazhat; a moduláris szervezés rugalmasságot nyújt a tárkapacitás kiterjesztésében.

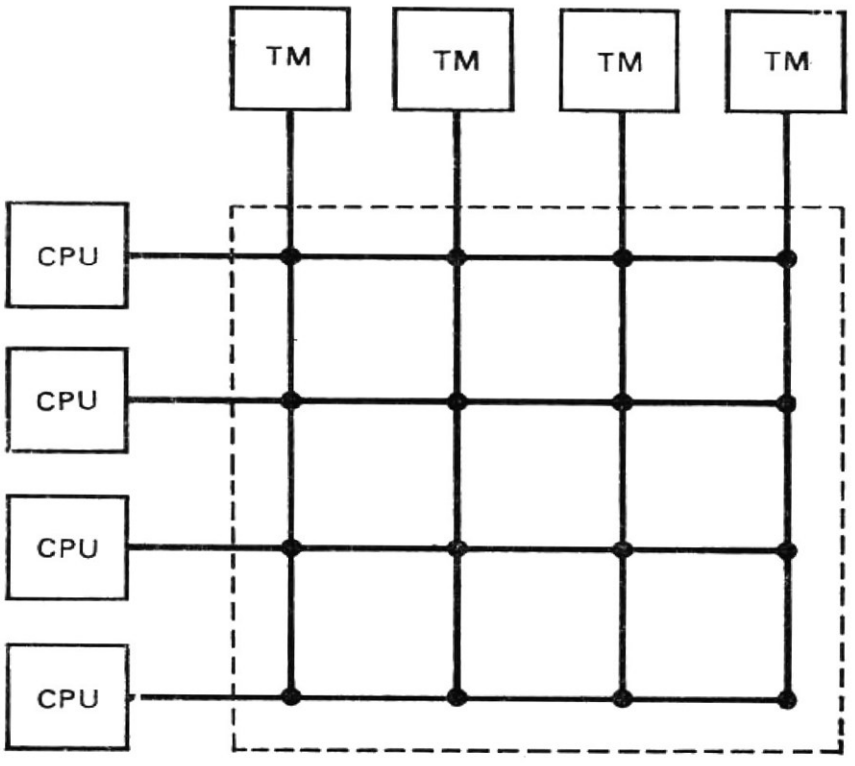
Ha a 7.2. ábrán látható tár négy moduljának tárciklusait a ciklusidő negyedével lépcsőzik, akkor — miután az első modul megkezdte saját tárciklusát, — a második, harmadik és negyedik modul a tárciklusát 1/4, 1/2, ill. 3/4 ciklusidővel később kezdi meg. Ha a társzavakat ilyen sorrendbeli elrendezésben érik el, akkor a maximális adatátviteli sebesség egy modul adatátviteli sebességének négyszeresét érhetné el. A valóságban a tárat véletlenszerűen érik el, és az átlagos adatátviteli sebesség valahol 1 és 4 között van. A tár előbb leírt működésmódját *beékelésnek* (interleaving) nevezik. A 7.2. ábrán levő tár kétutas vagy négyutas beékelést tesz lehetővé. Ha az utasításokat és az operandusokat két különböző modulban helyezik el, az utasításlehívó tárciklus és az operanduslehívó tárciklus beékelt lehet, ezáltal ténylegesen rövidebb tárciklusidőt hozhatunk létre; ezt gyakorta *átlapolásnak* (overlapping) nevezik. A következőkben leírásra kerülő, pufferszervezésű tárban a főtár négyutas beékelést alkalmaz, ezáltal lehetővé teszi, hogy egyetlen tárciklusidőben négy szóból álló blokkot lehessen elérni, és ily módon megnő a tárpufferezés hatékonysága.



7.2. ábra
A tármodul szervezése

7.1.3. Crossbarkapcsoló

A crossbarkapcsoló olyan kapcsolóberendezés, amely két modulcsoport tagjai között szimultán összekapcsolást tesz lehetővé. Elektromechanikus crossbarkapcsolókat már régen használnak az automata telefonközpontokban. A számítógépes alkalmazásokban a sebességigények miatt elektronikus crossbarkapcsolókra van szükség. A 7.3. ábrán 4×4-es elrendezésű crossbarkapcsolót láthatunk, ahol a négy tármodul (TM) közül bármelyik kommunikáltatható a négy processzor bármelyikével. Négy szimultán kommunikáció hozható létre. Minthogy a 7.3. ábrán látható crossbarkapcsoló mindegyik vonala igen sok vezetéket képvisel és a kapcsolásnak általában rendkívül gyorsnak kell lennie, egy 4×4-es elektronikus crossbarkapcsoló igen nagy rendszeregység. Bár a crossbarkapcsoló rugalmasságot jelent a tármodulok és a processzorok közti kapcsolatok létrehozásában, nem rugalmas viszont abban, hogy speciális alkalmazás esetén tetszőlegesen lehessen megválasztani a tármodulok vagy a processzorok számát, eltérően a crossbarkapcsolóhoz tervezett számtól. Más felépítésben a crossbarkapcsoló hardware-jét szétosztjuk a példánkban négynek választott tármodul között. Ekkor mindegyik tármodulnak négy terminálhalmaza van, amelyeket gyakorta *kimeneti kapocspárnak* (tail) vagy *kapunak* (port) neveznek, ez teszi lehetővé, hogy egy vagy több tármodulnak több processzorral is lehessen kapcsolata. Az elosztott vezérlésű megoldásban egy vagy több tármodult alkalmazhatunk, amelyek közül mindegyiknek egy vagy több processzora lehet; ez lehetővé teszi a rendszer lépcsőzetes kiterjesztését. Az is előfordulhat, hogy a tármodulnak különböző számú tárcsatlakozási kimeneti kapocspárja van, és hogy néhány tármodul egy vagy több processzor saját modulja, de nem tartozik más processzorhoz. Az elosztott megoldás előnye az, hogy az egyik kapcsolókomponens meghibásodása csak egyetlen egy tármodult tesz működésképtelenné és nem valamennyit.



7.3. ábra
Négy tármodult (TM) és négy processzort (CPU) összekötő crossbarkapcsoló

7.1.4.
Többszörös elérésű szervezés

A 7.1. és 7.2. ábrán bemutatott társzervezésen látható egy adatsín, amelyen az adatok a tárból, ill. a tárba vihetők. Egy adott időpillanatban csak egy adatszó (azaz átviteli egység) vihető át. Az ilyen társzervezést *egyszeres elérésű szervezésnek* nevezik. Ha a processzor és a csatornák (a csatornákat későbbi fejezetekben tárgyaljuk) egyidőben igényelnek tárhozzáférést, a tárciklust prioritásos alapon kell kiosztani.

Ha a társzervezés egynél több hozzáférést tesz lehetővé egyidőben, akkor *többszörös elérésű szervezésről* beszélünk. Ilyen megoldást mutat a 7.4. ábra. Az adatkapcsoló hálózatok kapcsolják a processzorból és a csatornákból jövő adatsíneket a négy tármodul tartalomregisztereihez, míg a címkapcsoló hálózatok a címsíneket kötik e négy modul négy címregiszteréhez. A szimultán hozzáférések szükségessé teszik, hogy ne legyen konfliktus akkor, ha ugyanazt a modult egynél több processzor és csatorna akarja elérni. Ha ilyen összeütközés előfordul, prioritásos hálózatokra van szükség (7.4. ábra). A négy adatsínnel az adatátviteli sebesség a négyszerese lehet a 7.1. ábrán látható, egyszeres elérésű szervezés átviteli sebességének. Ezek az adat- és címkapcsoló hálózatok, valamint a prioritásos hálózat hasonló funkciót valósítanak meg, mint az előbbiekben leírt, elosztott crossbarkapcsolók.

7.1.5.
A közvetlen elérésű tárak típusai

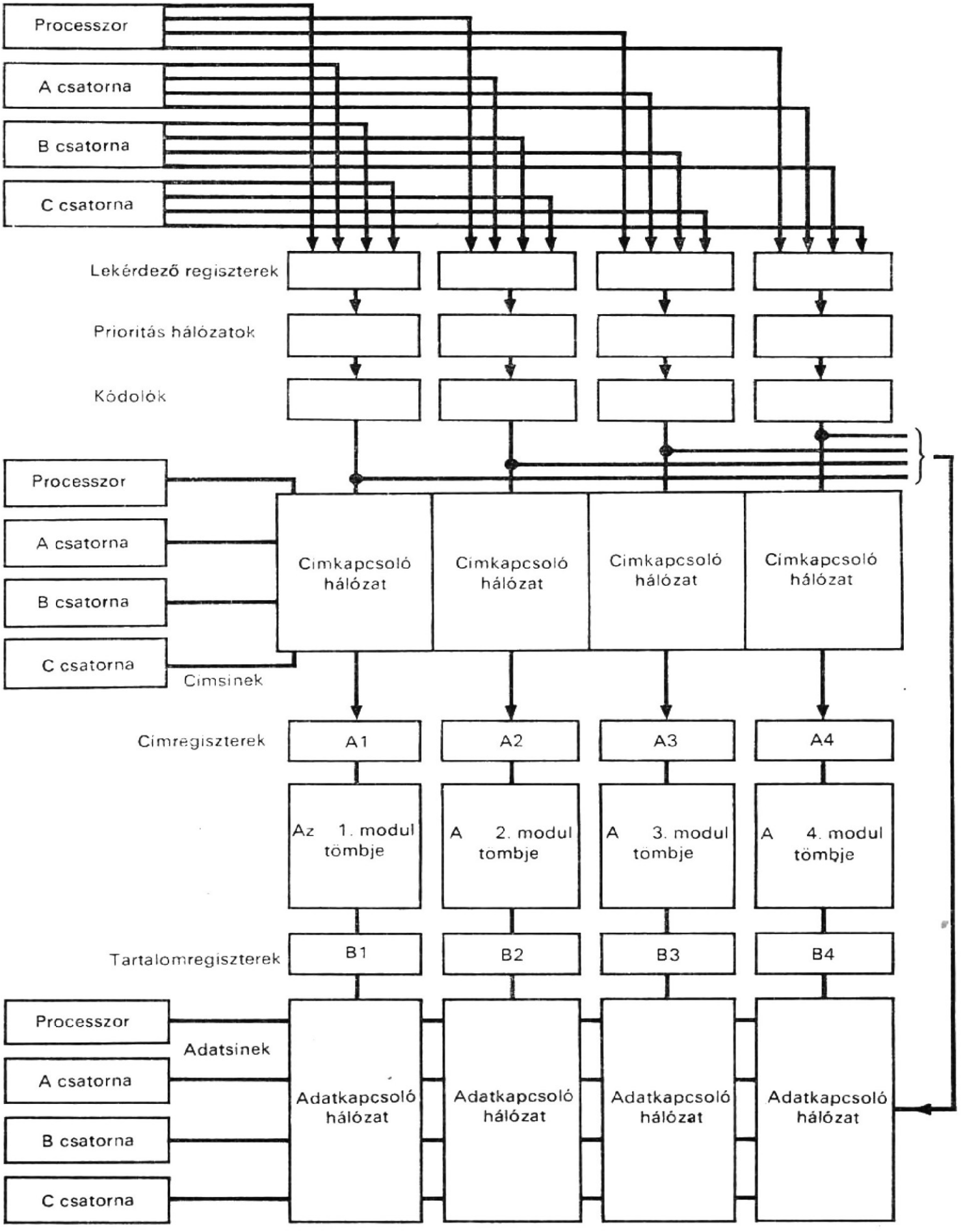
A közvetlen elérésű tárat a tár funkcionális vagy fizikai jellemzői szerint lehet osztályozni. A funkcionális jellemzők szerint van file-tároló, puffertároló, verem- (stack-) tár, programtár, mikroprogramtár, csatorna-programtár, csatornavezérlés tár stb. A fizikai jellemzők — sebesség és kapacitás — szerint van nagykapacitású tár, főtár, helyi tár, puffertár, vezérlőtár és regisztertömb; ezek hierarchiát alkotnak növekvő sebességű és csökkenő kapacitású tárakból. Vannak alkalmazások, ahol a főtár vagy a vezérlőtár mágneses vékonyréteg tár és a puffertár félvezető tár. A mai mikroprogramtárak többsége csak kiolvasható tár (Read Only Memory: ROM), amelyek rendszerint nem mágnesmagos tárak.

7.1. táblázat

IBM System/360 modellek tárjellemzői

Jellemzők	IBM System/360 modellek						
	30	40	50	65	75	85	91
Órajelgenerátor ciklusideje (μs)	0,5	0,625	0,5	0,2	0,195	0,08	0,06
Tárciklusidő (μs)	2,0	2,5	2,0	0,75	0,75	1,04	0,75
Tárelérési idő (μs)	1,0	1,0	1,0	0,6	0,585	0,88	0,60
Tárbeékelés	nincs	nincs	nincs	2-utas	2-4-utas	2-4-utas	8-16-utas
Adatátviteli útszélessége (byte)	1	2	4	8	8	16	8
Max. tárkapacitás (byte)*	65K	256K	256K	1024K	1024K	4096K	4096K
Min. tárkapacitás (byte)*	8K	16K	64K	128K	128K	512K	1024K

* A K 1024-gyel való szorzást jelent.



7.4. ábra
Modulszervezés többszörös eléréssel

Illusztrációképpen a 7.1. táblázat bemutatja az IBM 360-as rendszere 30, 40, 50, 65, 75, 85 és 91-es modelljeinek tárait (ezeket a jellemzőket az IBM a könyv megírásakor éppen módosítja). Ezek mágnesmagos tárak. A 7.1. táblázatban felsorolt jellemzők a következők: tárciklus és elérési idő, tárbeékelés, adatátviteli útszélesség, maximális és minimális tárkapacitás és óraciklusidő. Általánosságban, ahogy nő a modellszám, úgy csökken az órajelgenerátor ciklusideje és a tár elérési ideje (kivéve a 40-es modell órajelgenerátor ciklusidejét és tárciklusidejét és a 85-ös modell elérési idejét); növekszik viszont a beékelések száma; nő az adatátviteli út szélesség (kivéve a 91-es modellt); egyaránt nő a maximális és minimális kapacitás.

E modellek közül a 85-ös a legújabb. Bár hosszabb tárciklusidejű és hosszabb elérési idejű tárat alkalmaz (azonban nagyobb adatátviteli út szélességgel), mint amilyenek a 91-es modellnél vannak, nagysebességű puffert használ, amely rendkívül gyors, 80 ns ciklusidejű tár. A nagysebességű puffert felhasználó társzervezést a következőkben írjuk le.

7.2. Tárcímzés

Tárcímzésről akkor beszélünk, ha a tárban tárolt információ címezve van. Azt az információegységet, amelyet meg lehet címezni, *címezhető egységnek* neveznek. Címezhető egység lehet egy bit, byte, szó, félszó, egész szó, kétszeres szó, blokk, szektor vagy lap. A tár egy vagy több címegységének címzési lehetősége nagy jelentőségű a számítógép alkalmazásában.

A tárolt programú számítógép címzési rendszerének kiválasztásának szempontjai:

— bináris cím alkalmazása,
— a címezhető egység kiválasztása;
— igen nagy főtárcímzési lehetőség;
— a programok bithatékonysága;
— a betöltés és áthelyezés egyszerűsége;
— a dinamikus helyfoglalás és áthelyezhetőség megvalósíthatósága.

A bináris címnek más címek — pl. decimális cím helyett történő — alkalmazásával elkerülhető a konverzió. Ezért a jelenlegi közvetlen elérésű tárak címzésében a bináris cím a leggazdaságosabb hardware szempontból. Címezhető egységnek rendszerint egy byte-ot vagy egy szót választanak. A byte-nak mint címezhető egységnek a megválasztásával könnyebbé válik a szimbólumláncok (string) címzése, míg a szónak címezhető egységként való kiválasztása a fixpontos és lebegőpontos számok címzését teszi kényelmesebbé. Ha a blokkot választjuk címezhető egységnek, könnyebb lesz az adatok címzése a főtár és a külső tár (pl. egy mágneslemeztár) közötti átvitelhez.

Igen fontos jellemzővé vált a nagykapacitású tárak címzési képessége, mivel a nagykapacitású tár alkalmazása csökkenti a külső tárra való hivatkozások számát. A bitre való címzés több bitet igényel a tárcímben; kis felhasználási gyakorisága is ezt támasztja alá. A bithatékonyság a tárban tárolt bitek hatékony kihasználására utal. A hosszúcímű utasítások a tár jelentős részét foglalják el és így a hatékonyság kicsi lesz. A programoknak a számítógépbe való betöltése egyre bonyolultabb feladattá válik. A betöltés folyamán módosítani kell a programokban levő utasítások címét. A programok dinamikus helyfoglalása és áthelyezése a tárban gyakorta időosztásos (time-shared), multiprogramozott alkalmazásban történik. Ma már rendelkezésre állnak olyan címzési rendszerek, amelyek nem igényelnek címmódosítást a betöltés alatt,

és dinamikusan tudják elvégezni a programok helyfoglalását és áthelyezését; ezekhez a címzési módokhoz viszont járulékos hardware szükséges.

Ebben a fejezetben különböző címzési módokat írunk le, CDL procedurális változatának felhasználásával. Ezek a címzési módok a következők:

— közvetlen (immediate) címzés;
— direkt címzés;
— indirekt címzés;
— indexelt címzés;
— relatív címzés;
— báziscímzés;
— implicit címzés és
— regisztercímzés.

7.2.1. Közvetlen, direkt és indirekt címzés

A közvetlen címzésben az utasítás címmezeje magát az egyik operandust tartalmazza, nem pedig egy címet. Ez az operandus rendszerint egész konstans, nem lebegőpontos szám. Legyen adott egy M tár a CR címregiszterrel és D tartalomregiszterrel, amelyekhez a PSZ programszámláló és az AC akkumulátor járul. Ezzel a struktúrával a *közvetlen* címzés a következőképpen írható le:

```
Comment, közvetlen, direkt és indirekt címzéshez szükséges struktúra      (7.1)
Register,    CR(0—14),                  $címregiszter
             D(0—35),                   $tartalomregiszter
             PSZ(0—14),                 $programszámláló
             I,                         $indirekt címzés, jelzőbit
             AC(0—35),                  $akkumulátor
             OP(0—11),                  $műveleti- (op-) kód-regiszter
Subregister, D(ADDR)=D(21—35),          $az utasítás címmezeje
Memory,      M(CR)=M(0—32677, 0—35),
```

```
Comment, az utasítás lehívása                                             (7.2)
             CR←PSZ,                    $a következő utasítás címének átvitele
                                          a CR-be
             D←M(CR),                   $az utasítás kiolvasása a tárból D-be
```

```
Comment, közvetlen címzés
             AC←0;                      $az akkumulátor törlése
             AC(21—35)←D(ADDR),         $az operandus átvitele az AC-ba
```

Az előzőkben az akkumulátort először nullázzák és az operandust csak ezután viszik át az AC-ba, mivel az operandusnak most csak 15 bitje van.

Direkt címzéskor az utasítás címmezeje egy operanduscímet tartalmaz, ennek alapján az operandust le kell hívni a főtárból. A (7.1)-ben megadott struktúrával a direkt címzés a következőképpen illusztrálható:

```
Comment, az utasítás lehívása                                             (7.3)
             CR←PSZ;        $a következő utasítás címének átvitele a CR-be
             D←M(CR);       $a következő utasítás kiolvasása a tárból D-be
```

```
Comment, direkt címzés
        CR←D(ADDR); $az operanduscím átvitele a CR-be
        D←M(CR);    $operandus kiolvasása a tárból
        AC←D;       $az operandus átvitele az AC-ba
```

Az *indirekt* címzéskor az utasítást először lehívják a tárból, éppúgy, mint a direkt címzés esetén. Ennek az utasításnak a címmezeje azonban egy másik címet — (a cím címét) — tartalmazza, a cím címéről kell a műveletben részt vevő operandust lehívni. Ez a folyamat annyiszor ismétlődhet, ahányszor akarjuk. Pontosabban legyen az OP regiszterben a műveleti kód és az I regiszter tároljon egy bitet, amely, ha 1, akkor indirekt címzést jelöl. Ezekkel a járulékos elemekkel az indirekt címzést a következőképpen illusztrálhatjuk:

```
Comment, az utasítás lehívása                                    (7.4)
        CR←PSZ;     $a következő utasítás címének átvitele a CR-be
        D←M(CR);    $az utasítás kiolvasása a tárból a D-be
Comment, indirekt címzés
        OP←D(0—11);     $az op-kód tárolása az OP-ban
        I←D(12);        $az indirekt címzés jelzőbit-tárolása az I-ben
        CR←D(ADDR);     $a cím átvitele a CR-be
        D←M(CR);        $a megcímzett mező kiolvasása a tárból
/X/     IF (I=1) THEN (CR←D(ADDR)) ELSE (GOTO Y;)
        IF (I=1) THEN (D←M(CR));
        IF (I=1) THEN (I←D(12), GOTO X),
/Y/     AC←D.
```

A fentiekben a D(12) bit jelöli az indirekt címzést, ha 1 az értéke. Miután lehívtak egy tárrekeszt, további rekeszek hívása szükséges, ha az operandus D(12) bitjének 1 az értéke. Ha a D(12) bit 0, a lehívott rekesz a kívánt operandus.

Ha csak két operandushívás van megengedve, ezt a címzést *egyszintű* indirekt címzésnek nevezzük; ha ennél több van engedélyezve, ezt *többszintű* indirekt címzésnek nevezzük.

Összegezve, a közvetlen címzés nem igényel operandushívást a tárból. A direkt címzéshez egy operandushívás kell. Az indirekt címzéshez két, vagy több tárciklus szükséges.

7.2.2. Indexelt címzés

Az indexelt címzés indexregisztert alkalmaz, amelyben egy inkrementumot (vagy dekrementumot) tárolnak. Az indexelt címzéskor a tényleges címet ki kell számítani, majd ezt viszik át a címregiszterbe, az operandus lehívása végett. A *tényleges cím* a választott indexregiszter tartalmának és az utasításcímmező tartalmának az összege. Legyen az XR regisztertömbben három indexregiszter. Ezekkel a járulékos regiszterekkel együtt, az indexelt címzés a következőképpen illusztrálható.

```
Array-register, XR(1—3, 0—14),                                   (7.5)
Comment, az utasítás hívása
        CR←PSZ;     $a következő utasítás címének átvitele a CR-be
        D←M(CR);    $az utasítás kiolvasása a tárból a D-be
```

```
Comment, indexelt címzés
        IF (D(18—20)=0) THEN (CR←D(ADDR)),
        IF (D(18—20)=1) THEN (CR←D(ADDR) add XR(1,)),
        IF (D(18—20)=2) THEN (CR←D(ADDR) add XR(2,)),
        IF (D(18—20)=4) THEN (CR←D(ADDR) add XR(3,));
        D←M(CR);
        AC←D;
```

A D(18—20) bitek az indexmezőt tartalmazzák: ezek specifikálják az indexeléshez használandó idexregisztert. Ha az indexmező tartalma 0, a D(ADDR) részregiszter tartalmát átviszik a CR regiszterbe. Ha a D(18—20) bitek tartalma 1, 2 vagy 4, akkor az XR(1,), XR(2,) vagy XR(3,) indexregiszterek tartalmát hozzá kell adni a D(ADDR) részregiszter tartalmához, hogy megkapjuk a tényleges címet. Az indexelt címzés alkalmazásával az utasítások címe könnyen módosítható egy-egy hurokban és ez az utasítások ismételt felhasználását rendkívül kényelmessé teszi.

Indexelt címzést inkrementálással (dekrementálással) lehet megvalósítani. Ebben az esetben a tényleges cím kialakítása és a CR regiszterbe történő beírása után az indexregiszter tartalmát 1-gyel megnövelik (vagy csökkentik). Ez elősegíti az indexregiszternek hurokvezérlésre történő alkalmazását. Mindezeket az alábbi utasítások illusztrálják.

```
Comment, az utasítás hívása                                    (7.6)
        CR←PSZ;         $a következő utasítás címének átvitele a CR-be
        D←M(CR);        $az utasítás kiolvasása a tárból a D-be

Comment, indexelt címzés, inkrementálással
        IF (D(18—20)=0) THEN (CR←D(ADDR)),
        IF (D(18—20)=1) THEN (CR←D(ADDR) add XR(1,),
           XR(1,)←countup XR(1,)),
        IF (D(18—20)=2) THEN (CR←D(ADDR) add XR(2,),
           XR(2,)←countup XR(2,)),
        IF (D(18—20)=4) THEN (CR←D(ADDR) add XR(3,),
           XR(3,)←countup XR(3,));
        D←M(CR);
        AC←D.
```

Indexszelt címzést indirekt címzéssel összekapcsolva is lehet alkalmazni. Ebben az esetben lehetővé válik a pointer automatikus léptetése egy operandustáblázat sorrendi elemein.

Az indexelt címzés másik kiterjesztése a *többszörös indexelés*, amelyben a tényleges cím kialakítása érdekében a növekmény egynél nagyobb lehet. Többszörös indexelést lehet végezni, ha az indexmező olyan társzó címét kezeli, amely egynél több címet tartalmaz. Ezek a címek majd olyan társzavak címei lehetnek, amelyeket indexregiszterekként használnak.

7.2.3.
Relatív címzés

A relatív címzés egy olyan utasításnak az operanduscímét specifikálja, amely az utasításszámlálóban levő címhez — ahol a pillanatnyi utasításcímet tárolja — képest relatív. Az előbbi struktúrával a relatív címzést a következőképpen illusztrálhatjuk.

```
Comment, utasításhívás                                          (7.7)
         CR←PSZ;        $a következő utasítás címének átvitele a CR-be
         D←M(CR);       $az utasítás kiolvasása a tárból a D-be
Comment, relatív címzés
         IF (D(13)=1) THEN (CR←D(ADDR) add PSZ)
           ELSE (CR←D(ADDR));
         D←M(CR);
         AC←D;
```

A fentiekben a D(13) bit a relatív címzést jelöli, ha értéke 1. A cím a programszámláló tartalmának és az utasításcímmező tartalmának az összege.

7.2.4.
Báziscímzés

A báziscímzés egy utasítás operandusának címét a BCR báziscímregiszterben levő címhez viszonyítva határozza meg. Tehát ez olyan relatív címzés, amely a bázis címregiszterben levő címet veszi figyelembe az utasításszámlálóban levő cím helyett. Ezzel a struktúrával a báziscímzést a következőképpen illusztrálhatjuk.

```
Register,  BCR(0—14),     $báziscímregiszter                    (7.8)
Comment, utasításhívás
         CR←PSZ;        $a következő utasítás címének átvitele a CR-be
         D←M(CR);       $az utasítás kiolvasása a tárból a D-be
Comment, báziscímzés
         IF (D(14)=1) THEN (CR←D(ADDR) add BCR)
           ELSE (CR←D(ADDR));
         D←M(CR);
```

A fentiekben a D(14) bit jelöli a báziscímzést, ha értéke 1. A cím a báziscímregiszter tartalmának és az utasításcímmező tartalmának az összege.

7.2.5.
Regisztercímzés

Az operandusokat rendszerint a tárban tárolják. Néhány operandust (és néhány részeredményt) azonban gyakran az ún. általános regiszterek valamelyikében tárolnak, a folyamat gyorsítása érdekében. A regisztercímzés ilyen regiszterben levő operandus címzését végzi. Legyen a GR regisztertömb két, általános célú regiszter. A regisztercímzést a következőkben mutatjuk be.

```
Array-register,   GR(1—2, 0—35), $általános célú regiszterek          (7.9)
Register,         RA(0—1),       $a D(15—16) biteket tárolja
```

```
Comment, utasításhívás
                  CR←PSZ;        $a következő utasítás címének átvitele a CR-be
                  D←M(CR);       $az utasítás kiolvasása a tárból a D-be
Comment, regisztercímzés
                  CR←D(ADDR);    $a második operanduscím átvitele CR-be
                  RA←D(15—16);   $vezérlőbitek tárolása
                  D←M(CR);       $a második operandus hívása
                  IF (RA=2) THEN (GR(1,)←GR(1,) add D),
                  IF (RA=3) THEN (GR(2,)←GR(2,) add D);
```

Két operandus van: az egyik a GR(1,) vagy a GR(2,) regiszterben, a másik pedig a D(ADDR) regiszter által meghatározott tárcímen. A regisztercímzés bemutatása érdekében tételezzük fel, hogy összeadjuk ezt a két operandust. Amennyiben a D(15) bit értéke 1, a címzési mód: regisztercímzés. Ha a D(16) bit 0, a GR(1,) regiszter vesz részt a műveletben, egyébként pedig a GR(2,) regiszter.

Ha csak egyetlenegy általános rendeltetésű regiszter van és ez a regiszter az akkumulátor, nincs szükség az akkumulátor explicit címzésére. Az ilyen címzést *implicit címzésnek* nevezik.

Ha a fenti regiszterek operanduscímeket tartalmaznak operandusok helyett, ezt *indirekt regisztercímzésnek* nevezzük. Továbbá járulékosan még inkrementálás is lehet az indirekt regisztercímzés mellett. A fenti struktúrával, az inkrementálással ellátott indirekt regisztercímzést a következőképpen illusztrálhatjuk:

```
Comment, utasításhívás                                                 (7.10)
                  CR←PSZ;        $a következő utasítás címének átvitele a CR-be
                  D←M(CR);       $az utasítás kiolvasása a tárból a D-be
Comment, indirekt regisztercímzés, inkrementálással
                  IF (D(15—17)=1) THEN (CR←GR(1, 21—35),
                    GR(1, 21—35)←countup GR(1, 21—35)),
                  IF (D(16—17)=3) THEN (CR←GR(2, 21—35),
                    GR(2, 21—35)←countup GR(2, 21—35));
                  D←M(CR);
                  AC←D;
```

A fentiekben a D(17) bit az inkrementálásos indirekt regisztercímzést jelöli, ha értéke 1. A GR(1, 21—35) vagy a GR(2, 21—35) regiszter tartalmát (a címrészt) átviszik a CR címregiszterbe és a GR(1, 21—35) vagy GR(2, 21—35) regisztert 1-gyel megnövelik aszerint, hogy a D(16) bit értéke 0, ill. 1-e.

7.3. Stack- (verem-) tárak

A stack- (verem-) tár adatelemek lineáris listáját tartalmazza, a listába új elem bevitele betoldás (inzertálás), törlés lehet; az adatelemek elérése gyakran a lista egyik végén történik. Az ilyen típusú listát másképpen *push-down listának* vagy *last-in-first-out* (LIFO) *listának* is nevezik. A stack igen hatásos eszköz az egymásba ágyazott adatstruktúrák kezelésére.

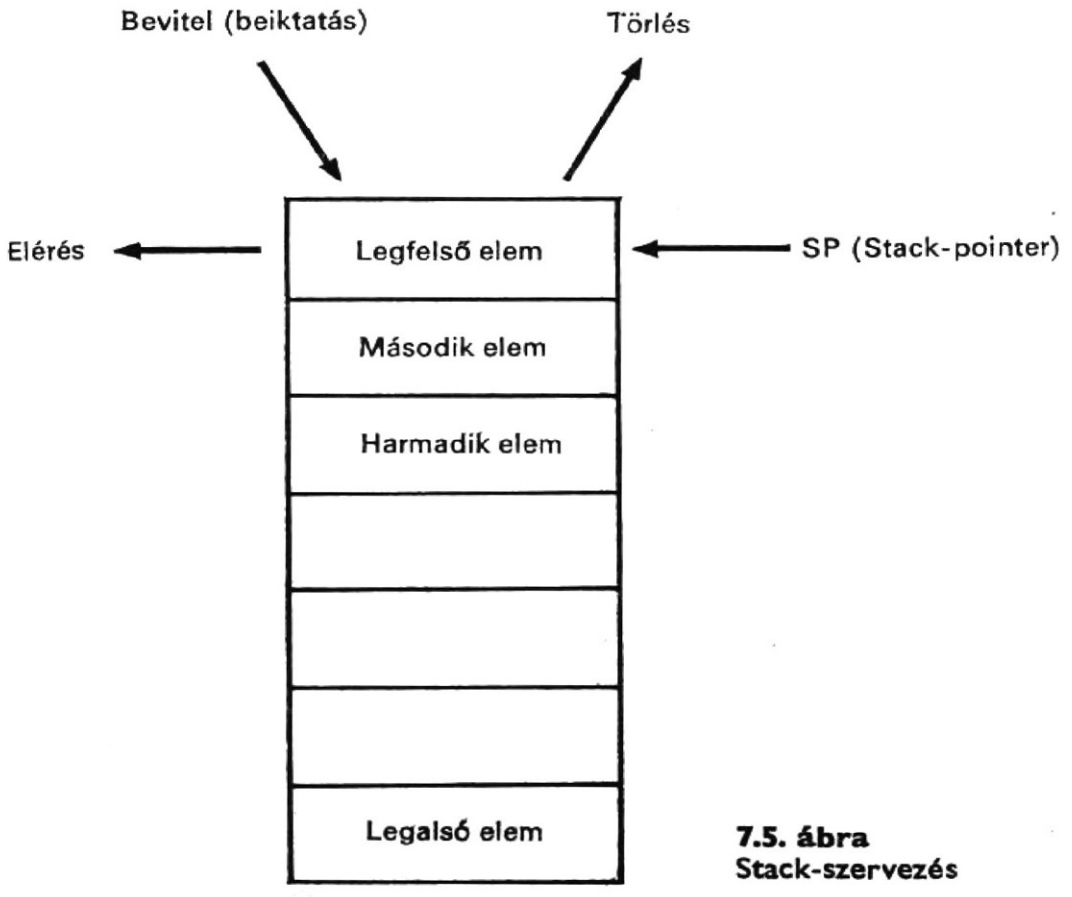
Más listák is előfordulnak, mint amilyenek a sorok és deque-ek. Vannak más adatstruktúrák is, így pl. a fa- és a többszörösen kapcsolódó struktúrák. Ezeket a struktúrákat hardware úton is meg lehet valósítani: az alábbiakban bemutatásra kerülő veremrealizálás példaként szolgál.

7.3.1. Stack (verem) szervezése

A hardware-ben megvalósított stack pl. a tárnak egy kijelölt területe. Stack-szervezésre a 7.5. ábrán látható példa, ahol a legfelső elemet, a második és harmadik elemet, majd a legalsó elemet tüntettük fel. Amint látható, a legfelső elemre mutat az SP *stack-pointer*. Legyen a tárban kijelölt terület 1024 szó, CR a címregiszter, TR a tartalom- (puffer-) regiszter és SP a stack-pointert tároló regiszter. A stack-szervezés konfigurációja az alábbiakban írható le.

```
Comment, stack-szervezés leírása                                    (7.11)
Register,  CR(0—14),          $tárcímregiszter
           SP(S, 0—9),        $a stack legfelső elemének pointere
           SM(0—14),          $a stack legalsó elemének pointere
           TR(0—35),          $tartalom- (puffer-) regiszter
           TÚLCSORDULÁS       $stack-túlcsordulásindikátor
           ALULCSORDULÁS      $stack-alulcsordulásindikátor
Memory,    M(CR)=M(0—32677, 0—35)
```

A fentiekben az SP(S) bitet használják a stack-alulcsordulás vagy stack-túlcsordulás jelzésére, amint azt a továbbiakban látni fogjuk.



7.5. ábra
Stack-szervezés

7.3.2.
Stack- (verem-) műveletek

Három fontos stack-művelet van:

— egy új elem beiktatása (inzertálás);
— egy elem törlése;
— egy elem elérése.

Ahhoz, hogy egy új elemet beiktassanak a stack tetejére, a legfelső elemet lenyomják, így második elem lesz belőle, és az új elemet helyezik el a stack tetején, így az legfelső elem lesz. Ahhoz, hogy egy elemet törölni lehessen a veremből, a legfelső elemet távolítják el és az eredetileg második elemből lesz a legfelső elem. Ahhoz, hogy a legfelső elemet el lehessen érni, a tartalmát átmásolják egy másik regiszterbe, míg az elem változatlan marad. Ezeket a stack-műveleteket írják le a következő utasítások:

```
Comment, a stack-műveletek leírása
Comment, push-down művelet (mélyítés)                          (7.12)
         SP←countup SP;
         IF (SP(S)=1) THEN (TÚLCSORDULÁS←1, GOTO X);
         CR←SM add 0—S(0—9);
         M(CR)←TR;
Comment, pop-up (felfelé mozgatás) művelet                     (7.13)
         CR←SM add 0—S(0—9);
         P←M(CR);
         SP←countdn SP;
         IF (SP(S)=1 THEN (ALULCSORDULÁS←1, GOTO X);
Comment, a legfelső elem elérése                               (7.14)
         CR←SM add 0—S(0—9);
         TR←M(CR);
/X/      END
```

A fenti push-down műveletben, amikor az SP pointer túllépi az 1023-at a countup mikroművelet után, az SP(S) bit 1 lesz és a stack túlcsordul. Hasonlóképpen a pop-up műveletnél, amikor az SP stack-pointer tartalma negatívvá válik egy countdn mikroművelet után, az SP(S) bit ismét 1 lesz és a stack alulcsordul.

7.3.3.
Stack- (verem-) beállítás

Egy utasítás végrehajtása során kívánatos lehet, hogy az akkumulátor legyen a stack legfelső eleme és a tár tartalom- (puffer-) regisztere a stack második eleme. Ennek megvalósítása végett — ezen utasítások végrehajtása előtt — a stack két legfelső elemét fel kell ugratni a tárból az A és a B regiszterekbe. Ezen utasítások végrehajtása után az A és B regiszterben levő két legfelső elem tartalmát mélyítve beírják a stack-be. Ezeket a műveleteket, amelyeket stack-beállításnak neveznek, hardware-rel realizálni lehet, és automatikusan mehet végbe. Példa erre a Burrough's B5500 számítógépének stack-tára [1].

A hardware megoldás illusztrálására a következőkben leírjuk a push-down stack-beállítását. Legyen R az a regiszter, amelyben a stack felső korlátja +1 tárolódik.

Használjuk az Y(1—2) regisztert a stack vezérlésére. Ha az Y(1) értéke 1, ez azt jelzi, hogy az A regiszter nem üres; ha az Y(2) egyenlő 1-gyel, ez azt indikálja, hogy a B regiszter nem üres. Legyen SP a stack-pointer, A az akkumulátor, CR a címregiszter, TR a tartalom- (puffer-) regiszter, TÚLCSORDULÁS a stack túlcsordulás-indikátora és M a tár. Ezt a struktúrát és a push-down műveletet az alábbiakban írjuk le.

```
Comment, a push-down stack-beállítási művelet leírása                    (7.15)
Register,  CR(0—14),          $címregiszter
           TR(0—47),          $tartalom- (puffer-) regiszter
           A(0—47),           $akkumulátor
           SP(0—14),          $stack-pointer
           R(0—14),           $stack-korlátpointer
           Y(1—2),            $stack-vezérlőbitek
           TÚLCSORDULÁS       $stack-túlcsordulásindikátor
Memory,    M(CR)=M(0—32677,0, 47),
/W/        IF (Y=0) THEN (GOTO W);
           IF (Y(2)=0 THEN (GOTO Z) ELSE (SP countup SP);
           IF (SP=R) THEN (TÚLCSORDULÁS←1, GOTO W);
           CR←SP;
           M(CR)←P, Y(2)←0;
/Z/        IF (Y(1)=0 THEN (GOTO W) ELSE (SP←countup SP);
           IF (SP=R) THEN (TÚLCSORDULÁS←1, GOTO W);
           TR←A, CR←SP;
           M(CR)←B, Y(1)←0, GOTO W;
```

Amint a fentiekben leírtuk, ha az Y regiszter értéke 0, az azt jelöli, hogy az A és B regiszterek tartalma mélyebbre került és a művelet befejeződött. Egyébként pedig, ha az Y(2) bit értéke 1, a B regiszter tartalmát kell mélyíteni a stack-tárban. Ebben az esetben az SP stack-pointert megnövelik 1-gyel és azután megvizsgálják, hogy vajon meghaladja-e az R regiszterben tárolt felső korlátot. Ha túllépi a felső korlátot, a TÚLCSORDULÁS regiszter 1 értéket kap. Ha nem lépi túl a felső korlátot, a B regiszter tartalmát mélyítik a stack-be és az Y(2) bit újból 0 értéket kap. Ekkor megvizsgálják az Y(1)-et 0-ra. Ha tartalma 0, a művelet befejeződött; egyébként pedig azt jelzi, hogy az A regiszter tartalmát kell beírni. Ebben az esetben az SP stack-pointert ismét megnövelik 1-gyel és újból megvizsgálják, hogy nem lépte-e túl a felső korlátot. Ha nem haladta meg a felső korlátot, az A regiszter tartalmát először átviszik a B regiszterbe és azután az információt mélyítve beírják a stack-be. Az Y(1) bitet visszaállítják 0-ra. Ezzel a művelet befejeződött.

7.4. Asszociatív tár

A közvetlen elérésű tárnak (RAM) az a tulajdonsága, hogy a tár csak tárolóeszköz és minden egyes tárban levő szó helyét egy cím határoz meg. Ezt a közvetlen elérésű tárat nevezhetjük *hely szerint címezhető tárnak* is. Az asszociatív tárnak — amellett, hogy tárolóeszköz is — az a sajátossága, hogy a benne levő szavak a tartalmuk szerint címezhetők és a tárnak párhuzamos keresési képessége is van. A tartalom szerinti címezhetőség azt jelenti, hogy a tárban levő szót (szavakat) egy adott kérdező (kereső) szó, az ún. *argumentum* szelektálható mezőjének egyeztetésével érik el, nem pedig cím szerint, mint a közvetlen elérésű tárakban (az asszociatív tár lehet hely szerint címez-

hető is). A párhuzamoskeresési képesség azt jelenti, hogy a keresőszót a tár összes szavával össze lehet hasonlítani. Ilyen meggondolásokból az asszociatív tárat *párhuzamos keresésű tár* vagy *tartalom szerint címezhető tár* néven ismerik.

Egyes asszociatív tárakban a tár valamennyi szavának valamennyi bitjét egyidejűleg (szimultán) hasonlítják össze egy argumentummal; ezt *szópárhuzamos keresésnek* nevezik. A szópárhuzamos keresés helyett az összes szó egyetlen bitjét is összehasonlíthatják szimultán módon a kérdezőszó megfelelő bitjével; ezt *bitpárhuzamos keresésnek* nevezik. A bitpárhuzamos keresés lassúbb a szópárhuzamos keresésnél, ha egyenlőségre (vagy egyenlőtlenségre) történő egyeztetési műveletet hajtanak végre. Másrészt a bitpárhuzamos keresésű asszociatív tár kevésbé költséges és a gyakorlatban általában jobban megfelel nagyobb kapacitású asszociatív tár céljaira.

7.4.1. Társzervezés

Az asszociatív tár szervezése a 7.6. ábra blokkdiagramján látható. A tárolótömb szószervezésű. A tárolótömbhöz van rendelve az A, M, TR, E, EE és EN regiszter:

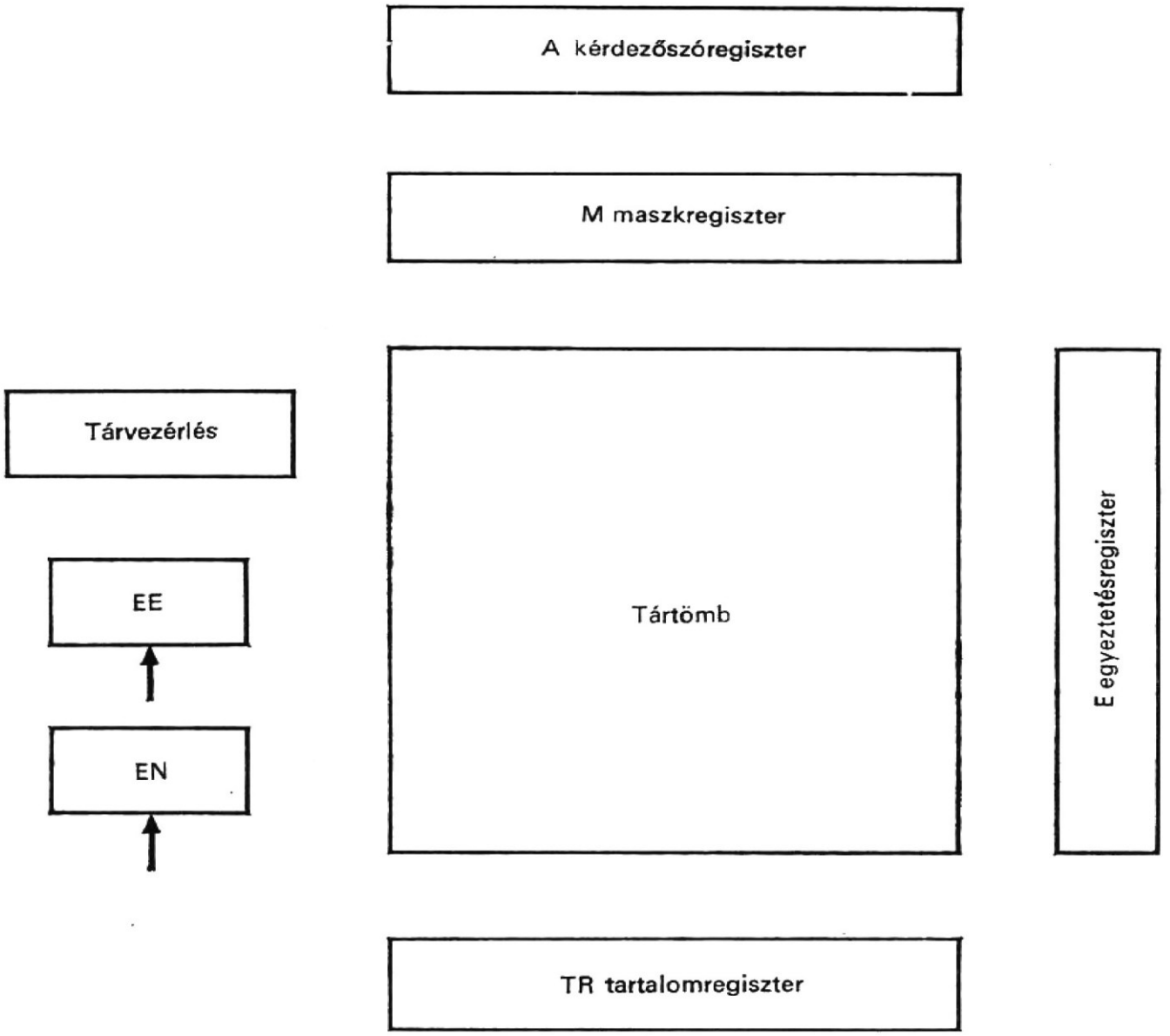
— az A regiszter a kérdezőszóregiszter;
— az M regiszter a maszkregiszter;
— a TR regiszter a tár tartalom- (puffer-) regisztere;
— az E regiszter az egyeztetést jelző regiszter;
— az EE és az EN regiszterek pedig az egyeztetés egyenlő feltételre, ill. egyeztetés nemegyenlő feltételre műveletek készítésére szolgálnak.

A 7.6. ábra szerinti asszociatív tárat a következő utasításokkal lehet leírni:

Comment, asszociatív tár leírása			(7.16)
Register,	A(0—35),	$kérdező szóregiszter	
	M(0—35),	$maszkregiszter	
	TR(0—35),	$tartalom- (puffer-) regiszter	
	E(0—255),	$egyeztetést jelentő regiszter	
	EE,	$egyeztetés—egyenlő feltételre vezérlő bit	
	EN,	$egyeztetés—nemegyenlő feltételre vezérlő bit	
Asso-memory,	MEM(A,M,TR)=MEM(0—255, 0—35),	$asszociatív tár	
Clock,	P,	$egyfázisú órajelgenerátor	
Terminal,	Z=M(0)+M(1)+...+M(35),		

Az egyeztetés — egyenlő feltételre művelet végrehajtásához a kérdező szót az A regiszterben és a maszkot az M regiszterben helyezik el. Ezután párhuzamos összehasonlítást végeznek egyenlőségre a kérdezőszó nemmaszkolt bitjei és a tárolótömb valamennyi szavában az ezeknek megfelelő bitek között. Azokat a szavakat, amelyek megegyeznek a maszkolt argumentummal, 1-gyel jelölik meg az E egyeztetést jelző regiszter megfelelő bitjeiben. Ezeket a mikroműveleteket a következő mikroutasításokkal lehet leírni.

```
/EE*P/  E(0)←((MEM(0,0)⊙A(0))+M(0)')*                                    (7.17)
              *((MEM)0,1)⊙A(1)(+M(1)')*
              *.......................*
              *((MEM(0,35)⊙A(35))+M(35)'*Z,
```



7.6. ábra
Asszociatív tár szervezése

$$
\begin{aligned}
E(1) \leftarrow & ((MEM(1,0) \odot A(0)(+ M(0)') * \\
& * ((MEM(1,1) \odot A(1)) + M(1)') * \\
& * \ldots\ldots\ldots\ldots\ldots\ldots * \\
& * ((MEM(1,35) \odot A(35)) + M(35)') * Z, \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
E(255) \leftarrow & ((MEM(255,0) \odot A(0)) + M(0)') * \\
& * ((MEM(255,1) \odot A(1)) + M(1)') * \\
& * \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots * \\
& * (MEM(35,35) \odot A(35)) + M(35)') * Z,
\end{aligned}
$$

Az előbbi mikroutasításokat meglehetősen hosszadalmas leírni. A következőképpen rövidíthetők le:

$$
\begin{aligned}
/EE * P/ \quad & E(0) \leftarrow * ((MEM(0, 0-35) \odot A(0-35)) + M(0-35)')) * Z, \\
& E(1) \leftarrow * ((MEM(1, 0-35) \odot A(0-35)) + M(0-35)')) * Z, \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
& E(255) \leftarrow * ((MEM(255, 0-35) \odot A(0-35)) + M(0-35)')) * Z,
\end{aligned}
\qquad (7.18)
$$

Figyeljük meg, hogy „*” (csillag) operátort helyeztük el közvetlenül a nyíl után, annak jelölésére, hogy ezt a műveletet annyiszor kell alkalmazni, amennyit az index mu-

t. A fenti mikroutasítást tovább rövidíthetjük, ha a „∗" operátort kétszer alkalmazzuk; így a következőket írhatjuk:

/EE∗P/ E(0—255)← ∗ ∗((MEM(0—255, 0—35) ⊙ A(0—35)+ M(0—35)′)∗Z. (7.19)

A fenti mikroutasítás azt mutatja, hogy az E egyeztetésregiszter egyes bitjei tárolják a párhuzamosan végzett összehasonlítási (egyeztetési) művelet azonossági jelzéseit. Az egyeztetés—egyenlő feltételre művelet hasonlóan megy végbe, kivéve, hogy az összehasonlítás végén az 1-esek az E megfeleltetésregiszterben a meg nem egyező szavakat jelölik. Természetesen, ha az előbbi összehasonlítás nulla, egy vagy több egyezést vagy megnemegyezést eredményezhet. Ha a kérdezőszó-regiszter minden bitpozícióját felhasználjuk a tárolótömb megfelelő bitjeivel való összehasonlításra, akkor ezt a tárat *teljesen asszociatívnak* nevezik; egyebekben pedig *részlegesen asszociatívnak*.

Ezeken az egyeztetés egyenlő feltételre vagy egyeztetés nemegyenlő feltételre műveleteken kívül az asszociatív tárnak még egy sor egyéb bázisművelete is van. Az *egyező szavak olvasása* és a *nemegyező szavak olvasása* műveletek kiolvassák a megegyező, ill. nemmegegyező szót vagy szavakat az asszociatív tárból a tartalom- (puffer-) regiszterbe. Az egyező szavak helyére történő írás és a nemegyező szavak helyére történő írás műveletek beírnak egy szót az egyező, ill. a nem pozíciókba; de az írás csak a szó nemmaszkolt mezőjére történik. A *törölj minden szót* művelet törli az összes szót az asszociatív tárból, beleértve az egyes szavak jelzőbitjeit is. (A kiegészítő jelzőbit azt jelöli, hogy a szó üres-e vagy a tár adott pozíciójába történt-e már írás.) A *jelölj meg minden bitet* művelet 1-eseket ír be az asszociatív tártömb összes nemmaszkolt bitpozícióiba. Az *írj a következő üres helyre* művelet egy szót ír be a tár következő üres helyére. (A következő üres helyet az első nullánál találjuk a társzavak első jelzőbitjében.) Az *egyező szavak számlálása* és a *nemegyező szavak számlálása* műveletek az 1-esek, ill. a 0-k számát határozza meg az E egyeztetési regiszterben. Továbbá van még két bitszámlálási művelet, amelyeket a következőkben írunk le.

7.4.2. Egyeztetés numerikus kérdezőszóra

Az egyeztetés numerikus kérdezőszóra műveletek párhuzamos összehasonlítást végeznek és ehhez az „egyenlőség", „nagyobb mint" vagy „határok között létezés" numerikus tulajdonságokat használnak fel. A következőkben hét ilyen műveletet írunk le. Ezek:

— egyeztetés nagyobb értékre;
— egyeztetés kisebb értékre;
— egyeztetés maximumra;
— egyeztetés minimumra;
— egyeztetés határok közötti értékre;
— egyeztetés legközelebbi nagyobb értékre;
— egyeztetés legközelebbi kisebb értékre műveletek.

A következőkben a „kérdezőszóbit" kifejezés mindig olyan kérdezőbitet fog jelenteni, amelyet nem maszkoltak le a maszkregiszterrel (vagyis a maszkbit 1).

Ezeknek az egyeztetés numerikus argumentumra műveleteknek a megvalósításához a 7.6. ábrán látható regiszterekhez képest még további regiszterek is szükségesek. Ezen regiszterek és egyéb számítógépelemek struktúráját az alábbiakban írjuk le.

Comment, asszociatív tár struktúrája (7.20)

Register,	A(0—35),	$a kérdezőszó-regiszter
	M(0—35),	$maszkregiszter
	TR(0—35),	$tartalom- (puffer-) regiszter
	E(0—255),	$egyeztetésregiszter
	M1(0—35),	$maszktároló regiszter
	M2(0—35),	$maszktároló regiszter
	M3(0—35),	$speciális maszkregiszter
	M4(0—35),	$speciális maszkregiszter
	A2(0—35),	$kérdezőszó tárregisztere
	SZ(0—5),	$számláló
	EE,	$egyeztetés egyenlőségre vezérlő bit
Asso-memory,	MEM(A,M,TR)=MEM(0—255, 0—35)	$asszociatív tár

A fentiekben az M1 és az M2 regiszter tárolja a maszkot, az A2 regiszter a kérdezőszót, az M3 és az M4 regiszterek pedig speciális maszkot tárolnak.

7.4.2.1. Egyeztetés nagyobb (kisebb) értékre

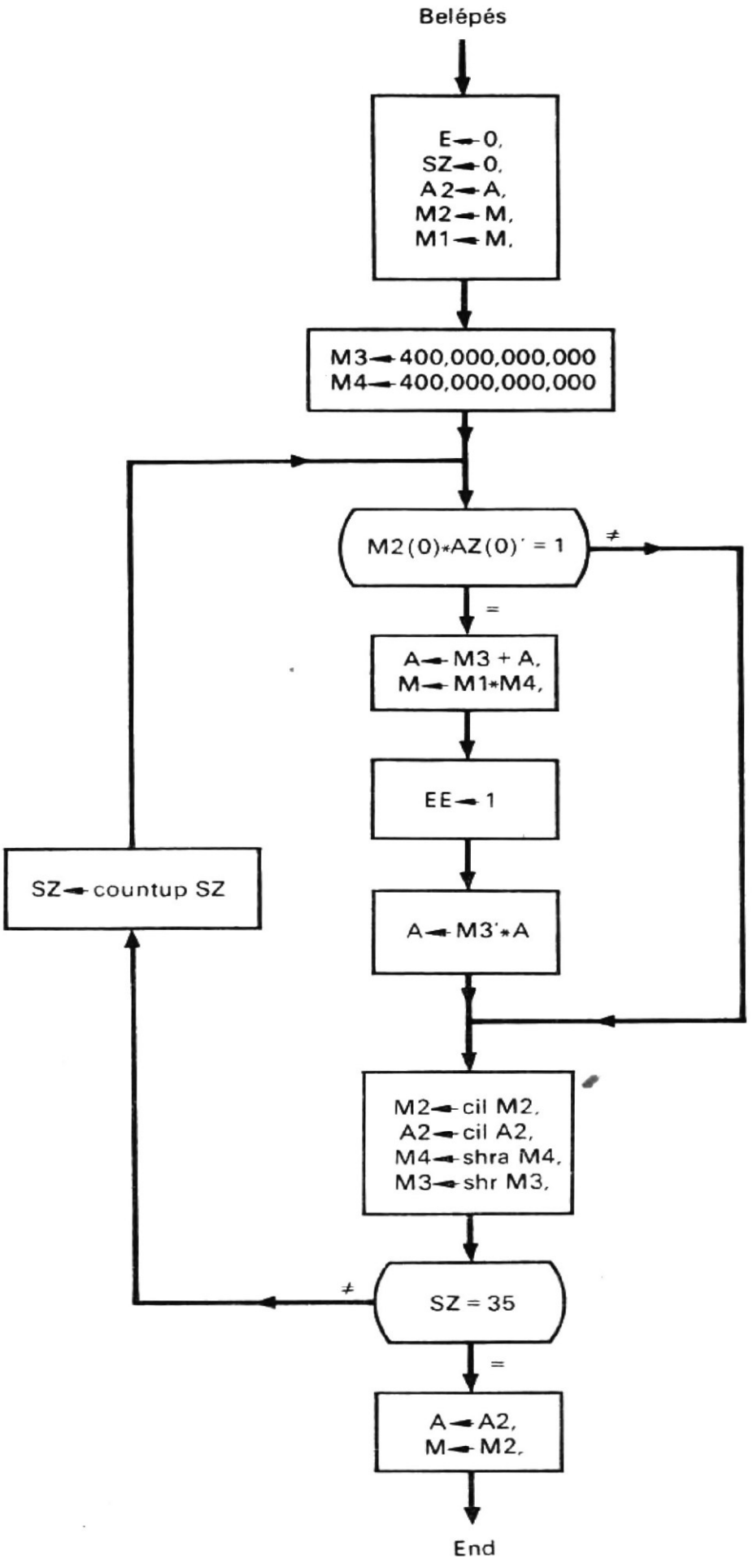
Az egyeztetés nagyobb (kisebb) értékre művelet azokat a tárbeli szavakat keresi ki, amelyek abszolút értéke nagyobb (kisebb), mint az A regiszterben levő kérdezőszó. A művelet algoritmusa következő:

a) Tapogassuk le a kérdezőszóbiteket a bal oldali legszélső bittel kezdve a jobb szélig, és azonosítsuk az első 0-t (1-est), amelyet *célpont-* (target-) *bitnek* nevezünk!
b) Képezzünk új kérdezőszót a célpontbitnek 1-gyel (0-val) való helyettesítésével!
c) Képezzünk új maszkot a célpontbittől jobbra eső valamennyi kérdezőszóbitnek a lemaszkolásával!
d) Végezzük el az egyeztetés egyenlőségre műveletet az új kérdezőszóra. (A megfelelteket az E regiszterben 1-esek jelölik és az E regiszter akkumulálja őket.)
e) Állítsuk vissza a célpontbitet a 0 (1) állapotba!
f) Ismételjük meg az *a)* lépést, a következő 0 (1) értéket célpontbitnek kiválasztva! Tartsuk meg a célpontbittől balra eső kérdezőszóbitet és maszkoljuk le a tőle jobbra eső kérdezőszóbiteket! Ismételjük meg az *a)...e)* lépéseket!
g) Ismételjük meg az *f)* lépést mindaddig, amíg több célpontbitet nem lehet meghatározni! Az argumentumnál nagyobb társzavakat most már az E egyeztetésregiszterben levő 1-esek határozzák meg!

Az ismertetett algoritmus folyamatábrája a 7.7. ábrán látható, ahol a kérdezőszót kezdetben az A2 regiszter tárolja, és a maszkot kiinduláskor az M1 és az M2 regiszterek tartalmazzák. Figyeljük meg, hogy az shr operátor logikai jobbra léptetést végez (azaz 0-t iktat be az üres bal oldali pozíciókba), míg az shra operátor aritmetikai jobbra léptetést hoz létre (vagyis a bal oldali bit változatlan marad). Az *a)* lépésben a célpontbitet a következő kifejezés identifikálja:

$M2(0) * A2(0)'$.

Az új célpontbitet úgy kapjuk meg, hogy ciklikusan balra léptetjük az M2 és az A2 regisztereket minden egyes iteráció után. A *b)* lépésben a célpont 0-ról 1-re változtatását a következő mikroművelet hajtja végre:



7.7. ábra
Folyamatábra egyeztetés-nagyobb-mint-kérdezőszó művelethez

A←M3+A,

ahol az M3 regiszter egy speciális maszkot tárol a célpontbit meghatározására. Ezt a maszkot minden egyes iteráció után logikailag jobbra léptetik. Az új maszkot a *c)* lépésben az M4 regiszterben tárolt speciális maszkkal alakítják ki, amelyik a szükséges maszkbiteket veszi ki az M1 regiszterből, amint azt a következő mikroművelet mutatja:

M←M1 * M4.

Az M4 regiszterben levő speciális maszkot minden egyes iteráció után aritmetikailag jobbra léptetik. Az egyeztetés egyenlőségre műveletet a *d)* lépésben az EE regiszternek 1 értékre állításával kezdik el, vagyis:

EE←1.

Az *e)* lépésben a célpontbitnek 1-ről 0-ra történő változtatását a következő mikroművelettel érik el:

A←M3' * A.

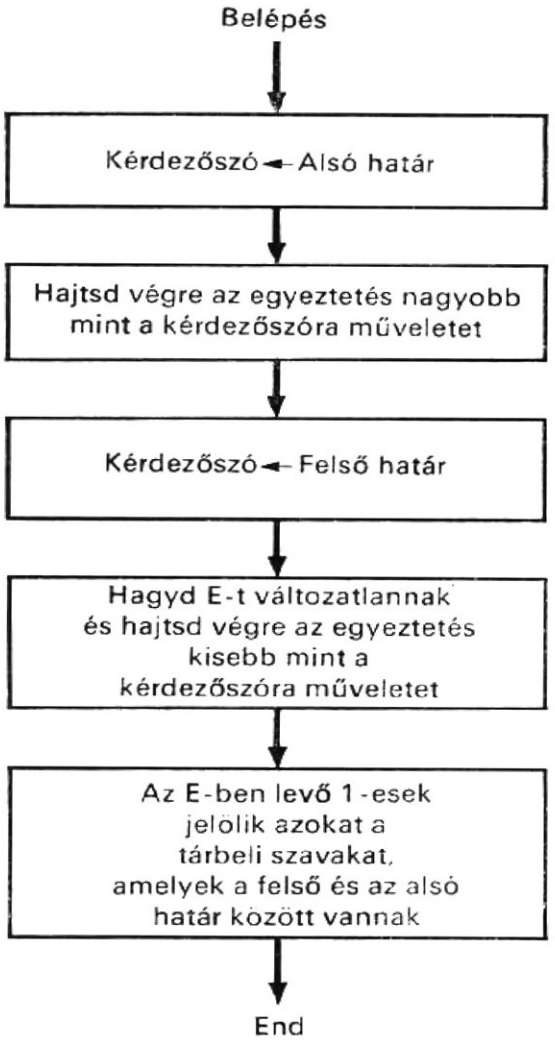
Ezt a folyamatot minden egyes célpontbitre megismétlik, amíg csak az SZ számláló el nem éri a 35 értéket. Ekkor a vizsgálati feltételt kielégítő szavak azok lesznek, amelyeket 1-esek jelölnek az E regiszterben, és az A és M regiszterekben a kérdezőszót visszaállítják.

Az egyeztetés nagyobb mint a kérdőszóra művelet sorrendjét a következő CDL procedurális leírással adjuk meg:

```
Comment, egyeztetés nagyobb mint a kérdőszóra sorozat                (7.21)
     SZ←0, E←0, A2←A, M2←M, M1←M,
     M4←400,000,000,000, M3←400,000,000,000;
/X/  IF (M2(0) * A2(0)'1) THEN (GOTO Y);
     A←M3+A, M←M1 * M4;
     EE←1;
     A←M3' * A;
/Y/  M2←cil M2, A2←cil A2 ,M4←shra M4,
     M3←shr M3;
     IF (SZ≠35) THEN (SZ←countup SZ, GOTO X);
     A←A2, M←M2;
     END
```

7.4.2.2. Egyeztetés határok közötti értékre

Az egyeztetés határok közötti értékre művelet azokat a tárbeli szavakat keresi ki, amelyek abszolút értéke két abszolútérték-határ, egy felső korlát és egy alsó korlát között helyezkednek el. Az algoritmust a 7.8. folyamatábra mutatja. Amint látható, két egyeztetési műveletre van szükség. Először az egyeztetésregisztert 0-ra kell állítani és egy egyeztetés nagyobb mint az argumentum műveletet kell elvégezni, ahol a kérdezőszó az alsó korlát. Azután az egyeztetésregiszter eredményeképpen kapott tartalmát kezdeti feltételként használva egy egyeztetés kisebb mint az argumentumra műveletet kell végrehajtani, ahol argumentum a felső korlát. E két művelet után az egyeztetésregiszterben levő 1-esek mutatják azokat a tárbeli szavakat, amelyek abszolút értéke a két korlát között van.

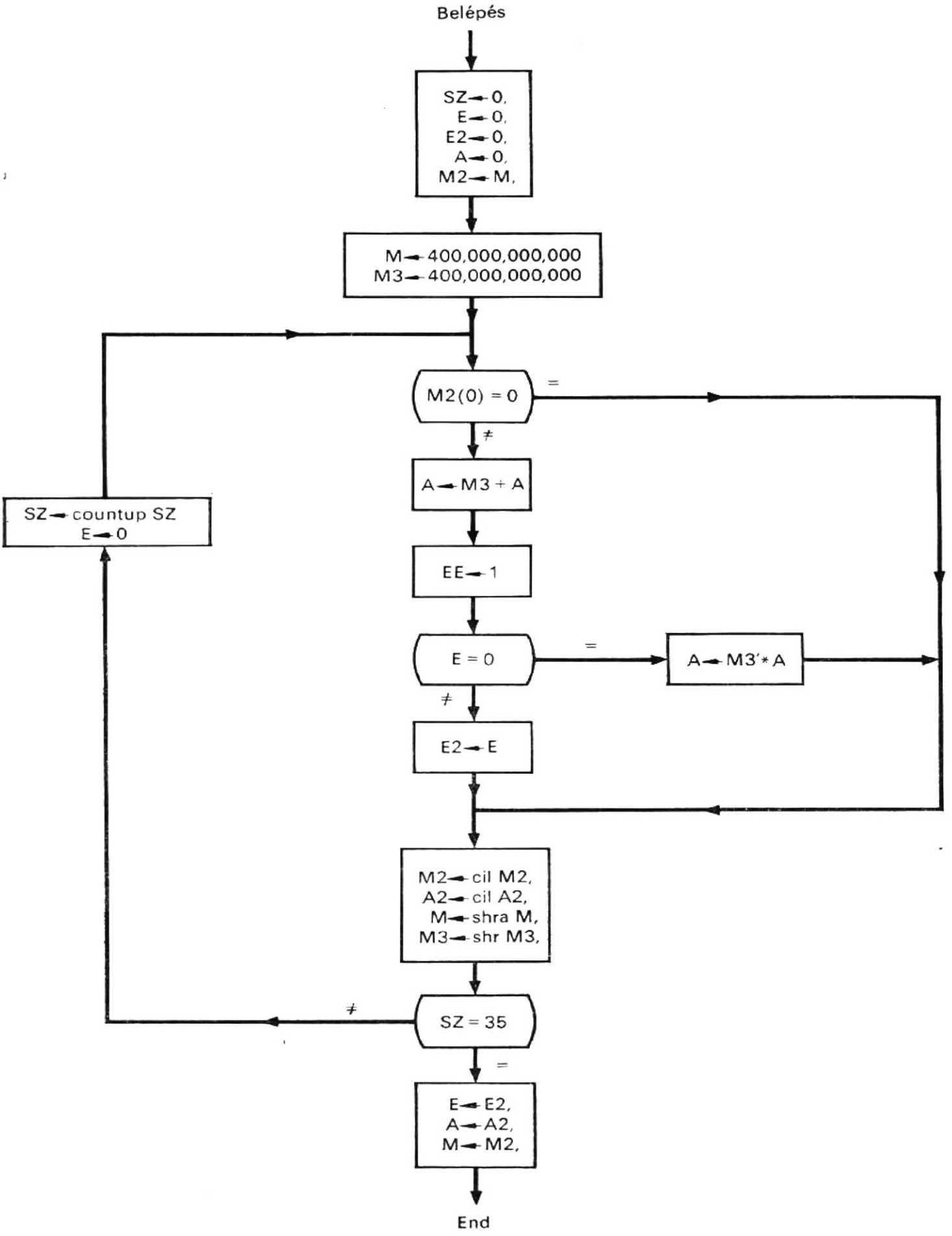


7.8. ábra
Az egyeztetés-határok-között művelet algoritmusát mutató hatásvázlat

7.4.2.3. Egyeztetés maximumra (minimumra)

Az egyeztetés maximumra (minimumra) művelet azokat a tárbeli szavakat keresi ki, amelyeknek numerikus maximumuk (minimumuk) van. A maszk az argumentum mezőjét jelöli, míg az argumentum maga nulla. Az algoritmust az alábbiakban írjuk le.

a) Tapogassuk le a kérdezőszóbiteket balról jobbra! Válasszuk a bal oldali legszélső kérdezőszóbitet *célpontbitnek*!
b) Képezzünk új kérdezőszót a célpontbitnek 1-re (0)-ra állításával és valamennyi többi, a célpontbittől jobbra álló kérdezőszóbitnek a lemaszkolásával!
c) Hajtsunk végre egyeztetés egyenlőségre műveletet az új kérdezőszóra!
d) Ha egy vagy több egyeztetés van, tároljuk az E egyeztetésregiszter tartalmát az E2 egyeztetés tárregiszterében. Ha nincs egyezés, ne tároljuk az E egyeztetésregiszter tartalmát, hanem állítsuk vissza a célponbitet 0-ra (1-re)!
e) Ismételjük meg az *a)* lépést a bal oldali legszélső kérdezőszóbitet célpontbitnek választva! Tartsuk meg a célpontbittől balra a kérdezőszóbiteket és maszkoljuk le a tőle jobbra eső kérdezőszóbiteket! Ismételjük meg a *b)*, *c)* és *d)* lépéseket!
f) Ismételjük az *e)* lépést mindaddig, amíg nincs több célpontbit! A maximum (minimum) az a tárbeli szó lesz, ahol 1-es van az E egyeztetésregiszterben. Vegyük észre, hogy egynél több tárbeli szó is lehet, amely ugyanazt a maximumot (minimumot) tartalmazza. Figyeljük meg, hogy a végső kérdezőszó megegyezik a maximummal (minimummal).



7.9. ábra
Folyamatábra a megfelelés-maximumra művelethez

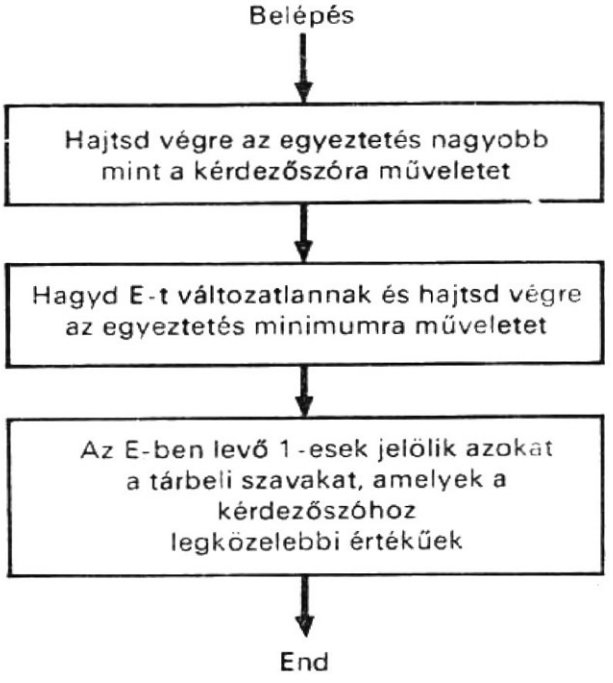
A fenti algoritmus folyamatábrája a 7.9. ábra. A kérdezőszó 0. A maszkot az M2 maszktároló regiszter tartalmazza. A célpontbitet az M2(0) bit határozza meg. Minthogy a 7.9. folyamatábra hasonló a 7.7. ábráéhoz, ezt a folyamatábrát nem részletezzük tovább.

Az egyeztetés maximumra sorrendet a CDL következő, procedurális változatával írjuk le.

```
Comment, megfeleltetés maximumra sorozat                    (7.22)
        SZ←0, E←0, E2←0, A←0, M2←M, M2←M;
        M←400,000,000,000, M3←400,000,000,000;
/X/     IF (M2(0)=0) THEN (GOTO Y);
        A←M3+A;
        EE←1;
        IF (E=0) THEN (A←M3'*A) ELSE (E2←E);
/Y/     M2←cil M2, A2←cil A2, M←shra M, M3←shr←M3;
        IF (SZ≠35) THEN (SZ←countup SZ, GOTO X);
        E←E2, A←A2, M←M2;
        END
```

7.4.2.4. Egyeztetés legközelebbi nagyobb (legközelebbi kisebb) értékre

Az egyeztetés legközelebbi nagyobb (kisebb) értékre művelet azokat a tárbeli szavakat keresi ki, amelyeknek értéke a legközelebbi nagyobb (kisebb) szám, mint egy adott szám. Az algoritmus folyamatábráját a 7.10. ábra mutatja. Amint látható, két műveletre van szükség. Először vissza kell állítani 0-ra az egyeztetésregisztert és az egyeztetés nagyobb (kisebb) értékre műveletet kell végrehajtani, az adott számot kérdezőszóként használva. Ezután az egyeztetésregiszterben eredményeképpen létrejövő tartalmat kezdeti feltételnek veszik, és egyeztetés minimumra (maximumra) műveletet hajtanak végre. E két művelet után az egyeztetésregiszterben levő 1-esek jelzik azt a tárbeli szót vagy szavakat, amelyek a legközelebbi nagyobb számot tartalmazzák, mint az adott szám.



7.10. ábra
Az egyeztetés-a-legközelebbi-nagyobb-értékre művelet algoritmusát mutató hatásvázlat

7.4.3.
Egyeztetés logikai (Boole-féle) kérdezőszóra

Az előzőekben leírt, egyeztetés egyenlőségre műveletben a kérdezőszóbitek és a nekik megfelelő bitek között a tárolótömbben történt párhuzamos összeállítás. Az egyenlőségre végzett összehasonlítás a logikai ekvivalencia műveletet használja fel. Az egyeztetés egyenlőtlenségre művelet párhuzamos összehasonlítás egyenlőtlenségre; ez az összehasonlítás a logikai KIZÁRÓ—VAGY műveletet használja fel. Ha más logikai műveletet, mint pl. ÉS, VAGY, NEM, NEMÉS, ill. VAGYNEM műveletet választunk, ezt *Boole-argumentumra való egyeztetésnek* nevezzük.

Ha ÉS műveletet választunk, az egyeztetések azokat a szavakat adják, amelyek 1-esei megfelelnek a kérdezőszóban levő 1-eseknek (de fordítva nem). Ha VAGY műveletet választunk, akkor az egyeztetések azokat a szavakat adják, amelyek 0-i megegyeznek a kérdezőszóban levő 0-kkal (de fordítva nem). Ha a NEM műveletet választjuk, akkor a nemegyezések azokat a szavakat adják, amelyek 0-i megegyeznek a kérdezőszóban levő 0-kkal (de fordítva nem).

Az előző egyezéseket (nemegyezéseket) *szimmetrikus egyezésnek* nevezik, mivel az egyezéshez egy 1 (0) bit szükséges a megfeleltetésregiszterben, és egy 1 (0) bit a kérdezőszóregiszterben. Ha az egyezéshez egy 1 (0) bit kell az egyeztetésregiszterben és egy 0 (1) bit az argumentumregiszterben, akkor azt *nem szimmetrikus egyezésnek* nevezik. Az ÉSNEM és a VAGYNEM műveletek nem szimmetrikus egyezéseket adnak.

7.4.4.
Egyeztetés számlálási értékre

Két bázisműveletet értelmeznek bitszámlálásra. A bitszámlálás művelet a tárban levő szó 1-eseit (vagy 0-it) számlálja (az éppen vizsgált szót vagy a tartalommező, vagy egy cím határozza meg, az utóbbi esetben a tárolóhely szerint is címezhető). A bitszámlálás és tárolás művelet a tárolás érdekében a számlálóban levő értéket átviszi az asszociatív tár tartalom- (puffer-) regiszterének egyik mezőjébe és azután a tartalomregiszter tartalmát elhelyezi az asszociatív tárban, az eredeti helyen. Minthogy a tárbeli szó mindegyik bitje egy attributumot ábrázolhat, az attributumok számának meghatározása hasznos kérdezőszó az attributumokban a zártság keresésére. Az egyeztetés számlálási értékre művelet a tárolótömb számlálási mezőjén végzett párhuzamos összehasonlításból áll.

7.4.4.1.
Egyeztetés bitszámra

Az 1-esek vagy 0-k alapján történő egyeztetések bitszámra művelet valamennyi tárbeli szó 1-eseit (vagy 0-it) leszámlálja és azután megkeresi az egyezéseket egy adott számmal. Ez a fajta egyeztetés az attributumok számának kiválasztásakor hasznos, amit az adott leszámolt érték reprezentál. Az 1-esek (0-k) leszámlálását és a számlálás eredményének tárolását az egyes tárbeli szavakban a bitszámlálás és tárolás művelet hajtja végre; az egyeztetést az egyeztetés egyenlő feltételre művelet végzi.

7.4.4.2.
Egyeztetés maximális (minimális) bitszámra

Az 1-esek (0-k) egyeztetése maximális (minimális) bitszámra művelet hasonló az egyeztetés bitszámra művelethez, azzal az eltéréssel, hogy a legtöbb 1-est (vagy 0-t) tartalmzó a tárbeli szót vagy szavakat keresi meg. Két műveletet használnak itt fel: az egyik a bitszámlálás és tárolás művelet, a másik pedig az egyeztetés maximumra (minimumra) művelet. A második művelet — kívánság szerint — lehet az egyeztetés a legközelebbi nagyobb (legközelebbi kisebb) értékre művelet is.

7.5.
Dinamikus betöltő

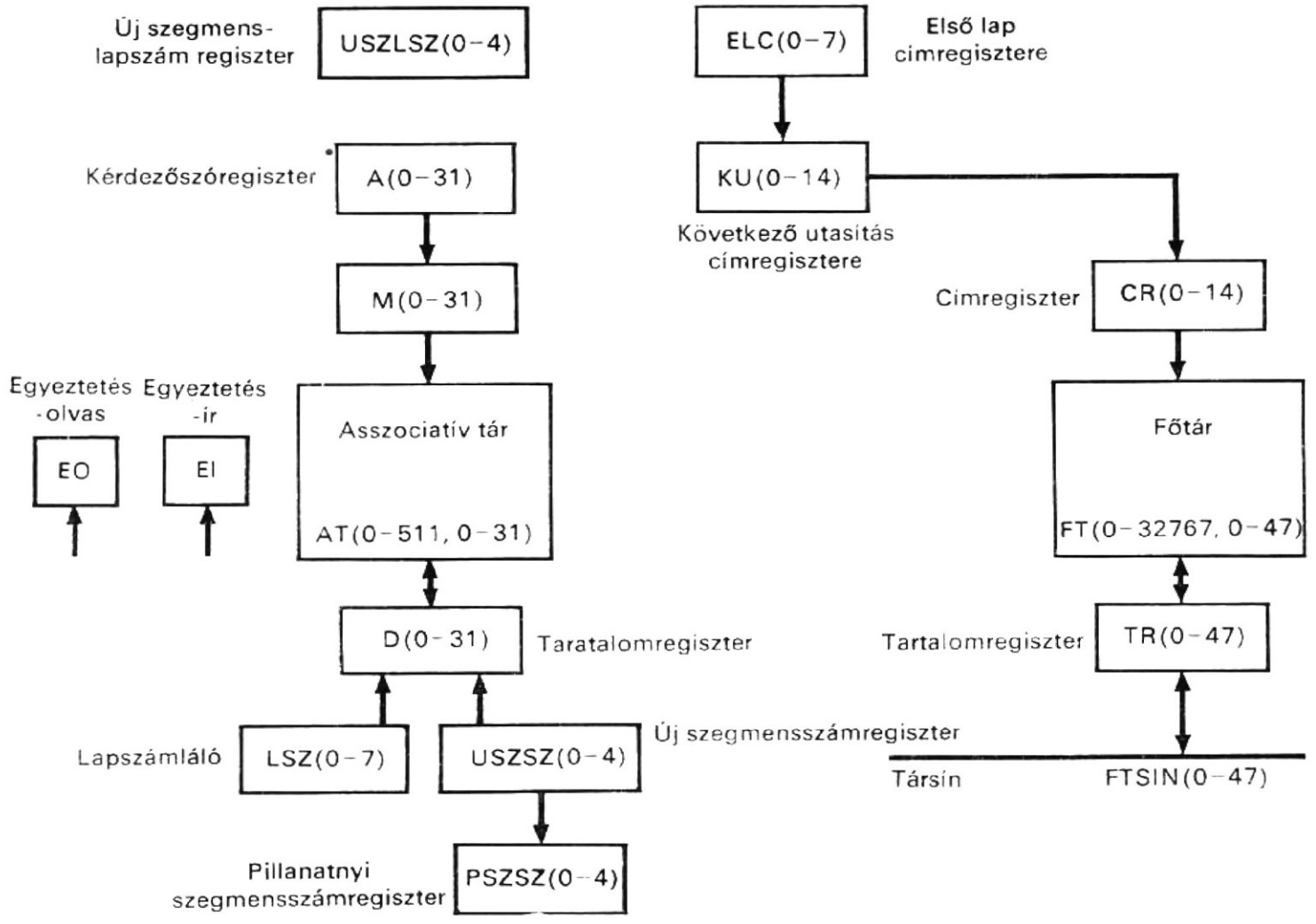
Ha egy programot vagy programszegmenset kell végrehajtani a processzorral, a program utasításait és adatait először be kell tölteni a főtárba. A betöltés előtt a betöltő (ismert angol nevén *loader*) egy-egy abszolút címet kell, hogy kijelöljön a program mindegyik szavához, és ha szükséges, megfelelő módon kell módosítania az utasításcímmezőben levő címet. Ha már egyszer be van töltve a program, nem célszerű a program újra történő áthelyezése (relokációja), a címmódosítás újabb feladatai miatt. Ez a fejezet egy olyan hardware betöltőt ír le, amelyben alkalmazott címzési eljáráshoz mindaddig nem szükséges címmódosítás, amíg a programot vagy programszegmenset valamilyen kiinduló ponthoz képest állították össze. Minthogy a betöltő a programot vagy programszegmenst bármelyik, a betöltési idő alatt rendelkezésre álló tárbeli helyre tölti be, *dinamikus betöltőnek* vagy még szemléletesebben *dinamikus allokáló betöltőnek* nevezik.

A dinamikus betöltőnek tudnia kell, mely tárolóhelyek állnak rendelkezésre allokációra és betöltésre; elég nagyok-e a helyek? Ha igen, hogyan lehet kényelmesen meghatározni és betölteni őket, és amellett még megvédeni a tárban levő egyéb programoktól, mivel a rendelkezésre álló helyek rendszerint inkább szétszórva találhatók, mint folytonosan. Tudnia kell, hogyan kezelje egy további programszegmens allokálását és betöltését, ha erre a szegmensre szükség van a végrehajtás során. A tárfelszabadítás, a szegmensmegosztás és a szegmensrelokáció is feladata. Ebben a fejezetben e feladatok közül néhánynak leírjuk a megoldását.

A következőkben feltételezzük, hogy a program egy vagy több szegmenset tartalmaz, és minden program vagy programszegmens kiindulópontját nullának tételezzük fel.

7.5.1.
A betöltő szervezése

Dinamikus betöltő szervezését a 7.11. ábra mutatja, ahol egy főtár, egy asszociatív tár, tizenhárom regiszter és egy társín található. A regiszterek a következők: CR címregiszter, TR tartalom- (puffer-) regiszter, KU következő utasítás címregisztere, PSZSZ pillanatnyi szegmensszám regiszter, ELC regiszter a rendelkezésre álló szegmens első lapcímének tárolására, USZSZ regiszter az új szegmens szegmensszámának tárolására, USZLSZ regiszter az új szegmens lapszámának tárolására, az LSZ lapszámláló, az A



7.11. ábra
Betöltő szervezése

argumentumregiszter, az M maszkregiszter, a D tartalom- (puffer-) regiszter, valamint az EO és EI vezérlésregiszterek. Az első két regiszter a FT főtárhoz van rendelve, az utolsó öt pedig az AT asszociatív tárhoz. A társín a TR tartalom- (puffer-) regiszterhez csatlakozik. Ezeket az elemeket a következő utasítások írják le:

Comment, dinamikus betöltő struktúrája (7.23)

Register,	CR(0—14),	$a főtár címregisztere
	TR(0—47),	$a főtár tartalom- (puffer-) regisztere
	KU(0—14),	$következő utasítás címregisztere
	PSZSZ(0—4),	$pillanatnyi szegmensszám regiszter
	ELC(0—7),	$első lap címregisztere
	LSZ(0—7),	$lapszámláló
	USZSZ(0—4),	$új szegmensszámregiszter
	USZLSZ(0—4),	$új szegmenslapszám regiszter
	A(0—31),	$kérdezőszóregiszter
	M(0—31),	$maszkregiszter
	D(0—31),	$tartalom- (puffer-) regiszter
	EO,	$egyeztetés-olvas vezérlőbit
	EI,	$egyeztetés-ír vezérlőbit
Memory,	FT(CR)=FT(0—32767,0—47)	$főtár
Bus,	FTSIN(0—47),	$főtársin

Asso-memory, AT(A,M,D)=AT(0—511, 0—31),$asszociatív tár
Subregister, A(X,Y,Z,SZSZ,S)=A(0—7,8—15,16—23,24—28,29—31),
M(X,Y,Z,SZSZ,S)=M(0—7,8—15,16—23,24—28,29—31),
D(X,Y,Z,SZSZ,S)=D(0—7,8—15,16—23,24—28,29—31),
KU(LC, SC)=KU(0—7, 8—14),
CR(LC,SC)=CR(0—7,8—14),

7.2. táblázat

A főtár jellemzői

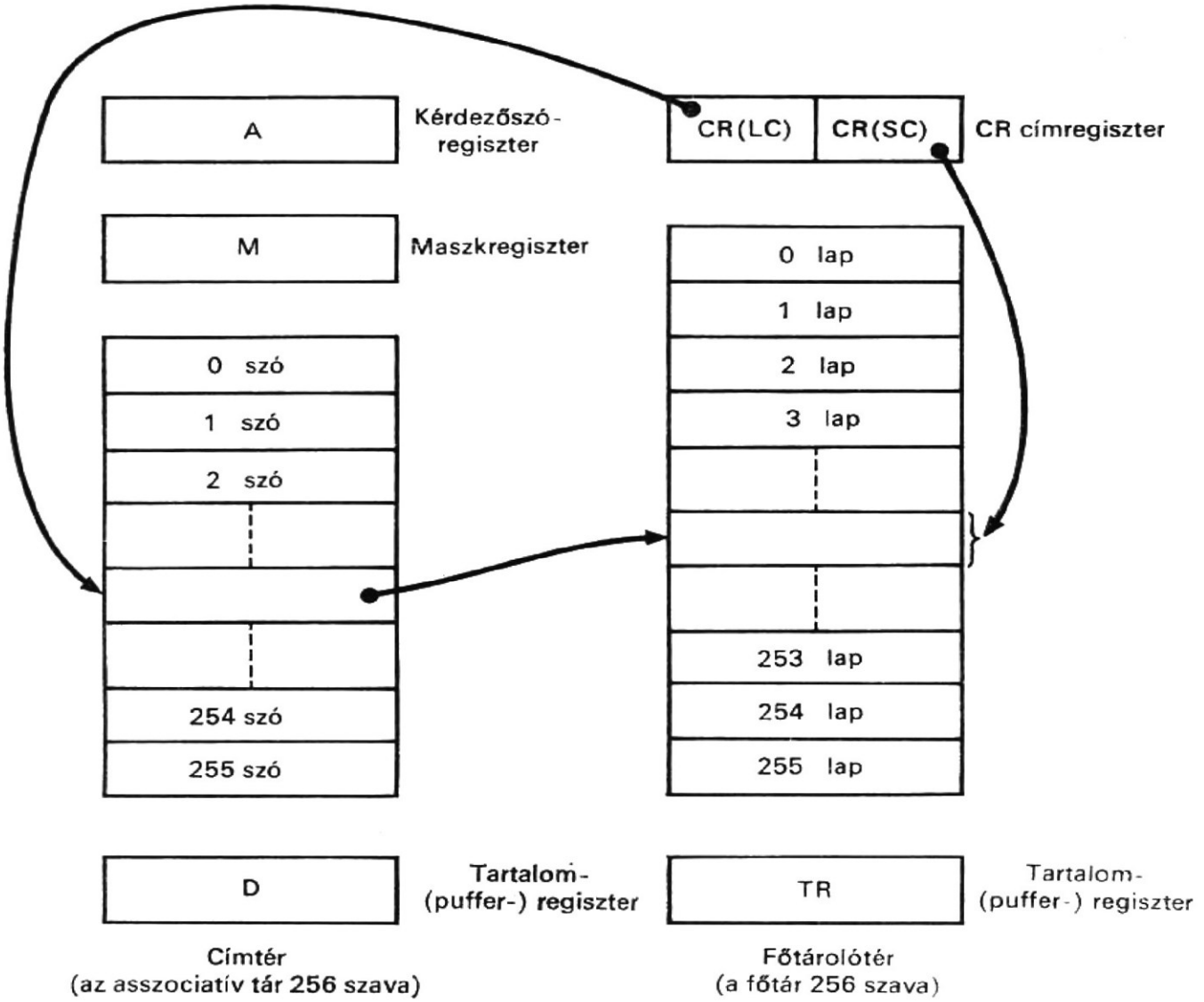
Jellemző	Érték
Szóhosszúság	48 bit
Lapméret	128 folytonosan következő szó
Kapacitás	256 lap vagy 32 768 szó
Lapcím	8 bit
Sorcím	7 bit
Teljes cím	15 bit
Ciklusidő	1 μs

A főtár hagyományos, kivéve, hogy lapokra van osztva. Az FT főtár jellemzőit a 7.2. táblázat tartalmazza. A főtárnak 256 lapja van, amelyek mindegyike 128 48-bites, folytonosan elhelyezett szót tartalmaz. A tárcímnek két része van: a 8-bites LC *lapcím* a 256 lap mindegyikéhez, és a 7-bites SC *sorcím* (a lapon belüli szó címe) a lapon elhelyezkedő 128 szó mindegyikéhez. A főtár lapjai és a főtár címregiszterének két része a 7.12. ábrán látható.

Az AT asszociatív tár jellemzőit a 7.3. táblázat foglalja össze. Az asszociatív tár 256 32-bites szót tartalmaz. Amint a 7.12. ábrán látható, mindegyik szó egy és csakis egy lapra mutat a főtárban. Minden szó öt mezőből áll; az egyes mezők az M maszkregiszterben levő maszkkal maszkolhatók. Az asszociatív tár mindegyik szava egyedi; ezért az egyeztetés során többszörös egyeztetés nem fordul elő. Az asszociatív tár két műveletet, egyeztetőszó olvasása és egyeztetőszó helyére írás műveletet tud végrehajtani. Az asszociatív tár ciklusideje 1/4-e a főtár ciklusidejének.

Az asszociatív tár szóformátumot a 7.13*a* ábra mutatja. A lapcímekre három, 8-bites mező áll rendelkezésre, az X, Y és Z; egy 5-bites mező a szegmensszámra; valamint egy 3-bites mező a státusz jelzésére. Az X lapcím valamelyik lap helyét mutatja a főtárban, az Y lapcím pedig a következő lap helyét. A 256 pár X és Y lapcím segítségével a főtár 256 lapját szegmensbe lehet összefogni. Az 5-bites szegmensszám segítségével a 256 lapot összesen 32 szegmensbe lehet összekapcsolni. A Z lapcím a lap elhelyezkedését mutatja a szegmensben, operandus címének hívásában való alkalmazásra, amint azt a későbbiekben le fogjuk írni. Az állapotmező a lap állapotát jelzi, az állapot rendeltetésére a 7.4. táblázatban található egy példa.

Példaként asszociatív tár leképzés látható a 7.13*b* ábrán, ahol a főtárnak csak 12 lapja van. A P1 programot (vagy egy program P1 szegmensét) három lapra töltötték be, amelyek lapcímei 1, 5 és 9; ezeket a lapokat az 1–5–9 összekapcsolt listába kapcsolják össze, és az 1 szegmensszámmal látják el. A P1 program első lapját az 1 lapcímen helyezik el. A szegmensszámot, az első lap címét és a program P1 nevét az asszociatív tárban tárolják, amint az a 7.13*b* ábrán az asszociatív tár leképezésnek a másodiktól az utolsóig terjedő soraiban látható. A 9. lapcímen levő lap az 1. számú szegmens utolsó lapja; ezt a csillag jelöli a 10-ik sor Y lapcímmezőjében, a 7.13*b* ábrán. Hasonlóképpen a P2, P3 és P4 programok sorra a 6–4–8, 3–11–0 és 2–10–7 összekapcsolt laplistákat tartalmazzák, 2, 3 és 4 szegmensszámokkal. A 7.4. táblázatban található szegmensállapot-kijelölés segítségével



7.12. ábra
A címtér leképezése a tár-térre

7.3. táblázat

Az asszociatív tár jellemzői

Jellemző	Leírás
Szóhosszúság	32 bit
Kapacitás	256 szó
Asszociativitás	teljesen asszociatív és maszkolható
Egyeztetés	csak egyetlen egyeztetés
Műveletek	*a)* egyeztetés-olvas
	b) egyeztetés-ír
Ciklusidő	1/4 μs egyeztetés-olvas vagy ír műveletre

— az 1. számú szegmens betöltött szegmens;
— a 2. számú szegmens szabad és készen áll arra, hogy rendelkezésre álló szegmens része legyen belőle;
— a 3. számú szegmens állandóan betöltve van (pl. a felügyelőprogram rezidens része számára);
— a 4. számú szegmens tartalmaz betöltésre rendelkezésre álló lapokat (ezt *rendelkezésre álló szegmensnek* nevezik).

X lapcím	Y lapcím	Z lapcím	Szegmens-szám	Állapot
8 bit	8 bit	8 bit	5 bit	3 bit

a)

X lapcím	Y lapcím	Z lapcím	Szegmens-szám	Állapot	
0	*	2	3	011	
1	5	0	1	000	
2	10	0	4	100	
3	11	0	3	011	
4	8	1	2	001	
5	9	1	1	000	
6	4	0	2	001	
7	*	2	4	100	
8	*	2	2	001	
9	*	2	1	000	
10	7	1	4	100	
11	0	1	3	011	
1	P1		1	110	Szegmensszám regiszter
6	P2		2	110	
3	P3		3	110	
2	P4		4	110	

b)

7.13. ábra
Példa asszociatívtár-térképre
a) szóformátum; *b)* tártérkép

7.4. táblázat

Példa állapot meghatározásra

Állapotmező	Állapot
000	betöltve
001	szabad
010	tartalékolt
011	állandóan betöltve
100	betöltésre rendelkezésre áll
101	megosztott
110	szegmensszám-regiszter

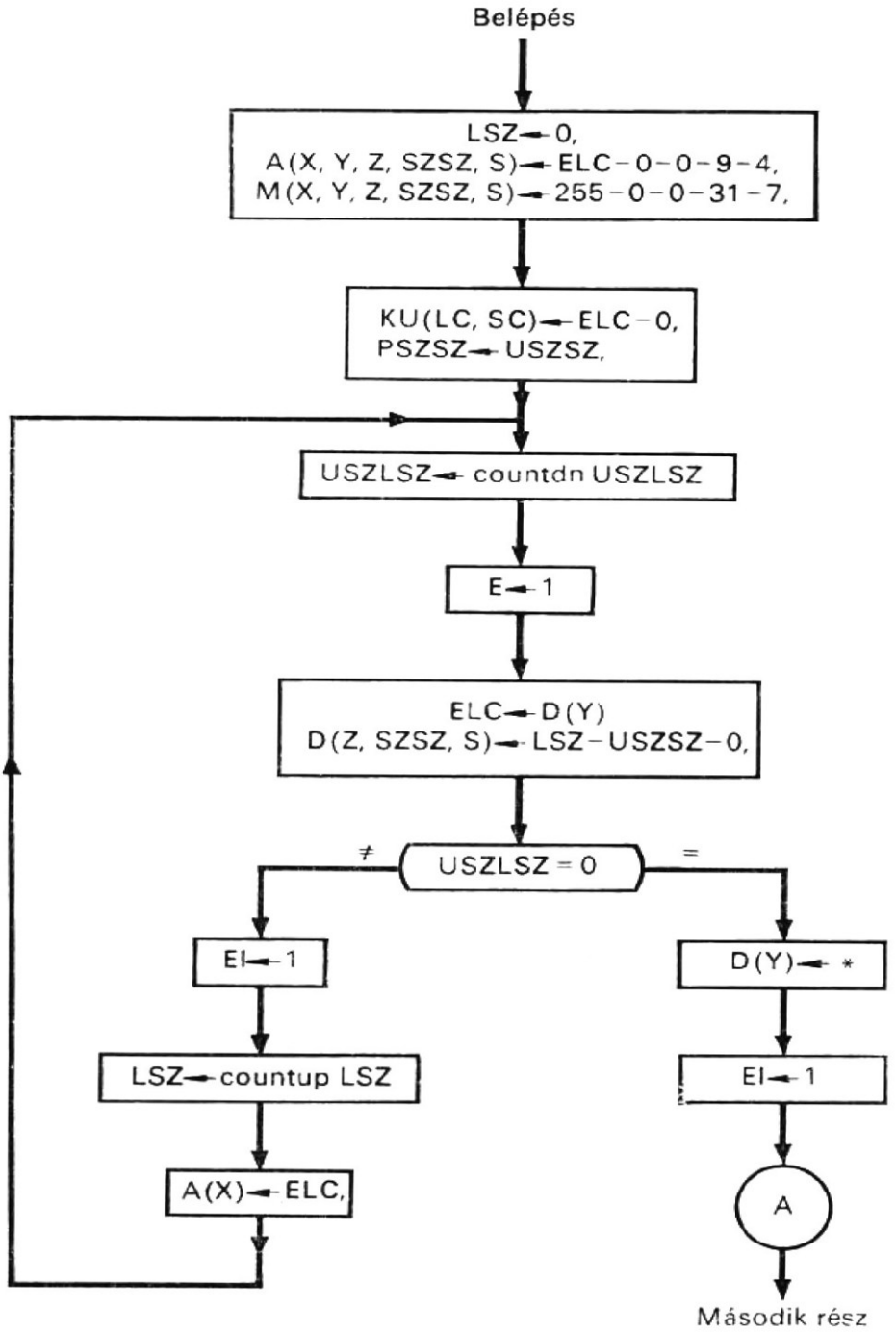
Az utolsó négy sor az asszociatív tár leképzésben a 7.13*b* ábrán azokat az asszociatív társzavakat ábrázolja, amelyek *szegmensszám-regiszterként* funkcionálnak, amit a 110 jelöl a státuszmezőjükben. Amint említettük, a szegmensek első lapcímeit a címek X mezői tárolják. A lapcímek Y és Z mezőit a visszatérési címek tárolására használják, amit a későbbiekben írunk le. A szegmensszám és az állapotmezők a szegmensszámot és a szegmensállapotot tárolják, amint azt az előzőkben láttuk.

Megjegyezzük, hogy a szavak sorrendje a 7.13*b* ábrán nem szükségszerűen a tényleges sorrend az asszociatív tárban; a tényleges sorrend valójában tetszőleges lehet.

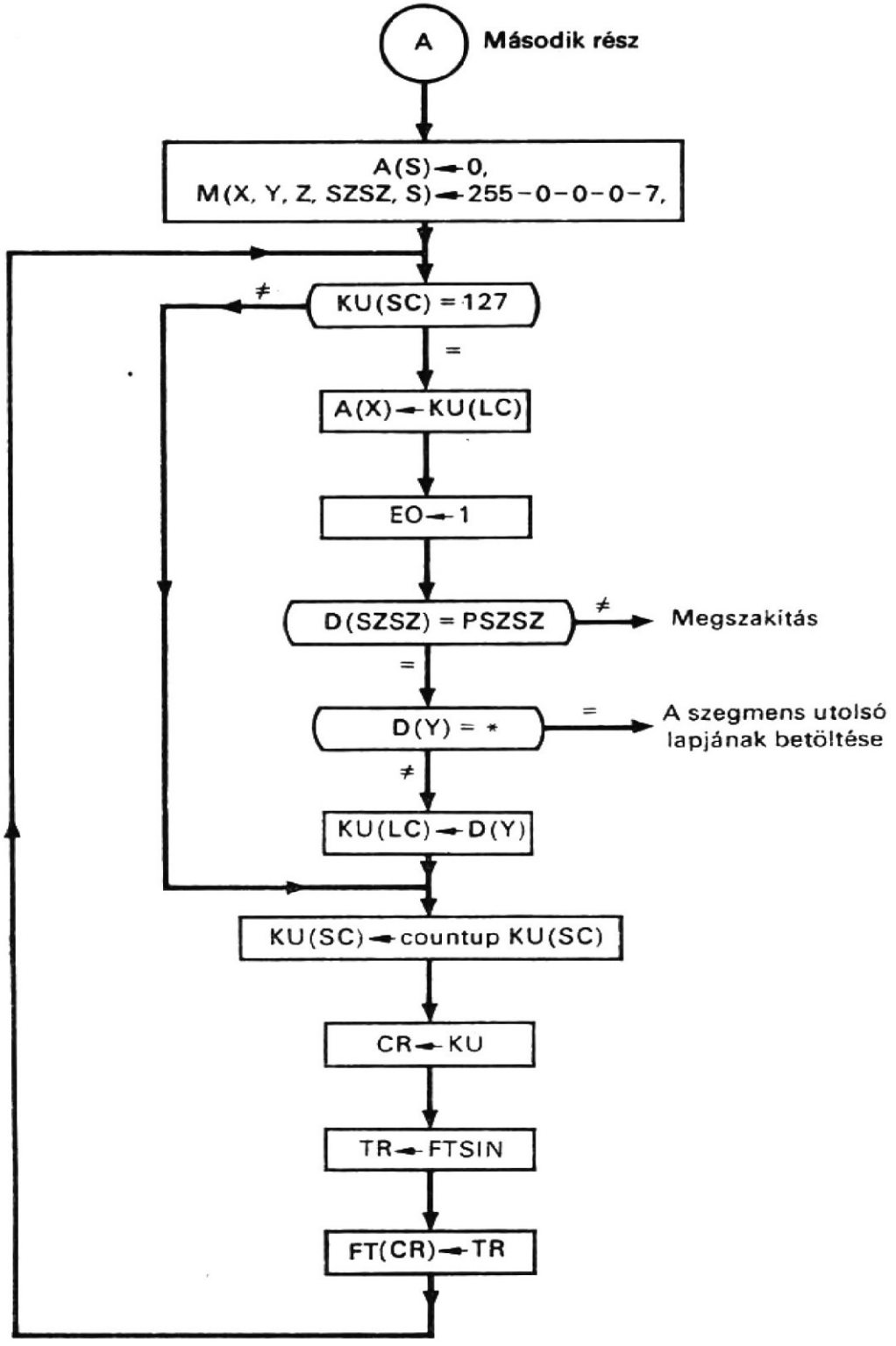
7.5.2.
Allokáció és betöltés

Mielőtt a betöltő elindítaná egy programszegmensnek a főtárba való betöltését, a felügyelőprogram meghatározza, hogy a programszegmens mérete kisebb-e, mint a rendelkezésre álló szegmensméret, és azután tárolja be az új szegmens lapjainak számát az USZLSZ új szegmenslapszám regiszterbe, valamint a hozzárendelt szegmenssz̈ámot az USZSZ új szegmensszám regiszterbe. A rendelkezésre álló szegmens első lapcíme az ELC első lapcím regiszterben van.

A betöltési folyamat a 7.14. ábrán látható. A sorozat azzal indul, hogy az LSZ lapszámlálót kinullázzák és beállítanak egy kérdezőszót az A regiszterbe és egy maszkot az M regiszterbe. Az argumentum az ELC regiszterben levő címet az X lapcímenként tartalmazza, valamint a 9 értéket a rendelkezésre álló szegmens példánkban választott



7.14. ábra
Dinamikus betöltő folyamatábrája



7.14/a ábra

szegmensszámaként és a 4 értéket állapotkijelölésként. (Betöltésre rendelkezésre áll). A maszkot úgy állítják be, hogy az Y és Z lapcímek ki legyenek maszkolva. A KU következő utasításcím-regiszter úgy kap kezdeti értéket, hogy az ELC regiszterben levő érték lesz a lapcím és a 0 értékű a sorcím. A PSZSZ pillanatnyi szegmensszám regiszter kezdeti értékét úgy kapja, hogy az USZSZ új szegmensszám regiszterben levő értéket veszik aktuális szegmensszámnak. Ezt a két regisztert, a KU és a PSZSZ regisztert inicializálják a betöltési folyamat második részéhez.

Mármost az allokálás első része az USZLSZ regiszter tartalmának 1-gyel való csökkentésével kezdődik. Az AT asszociatív tár a rendelkezésre álló szegmens első szavára egyezőszó-olvasás műveletet végez, az asszociatív tárból a D tartalom- (puffer-) regiszterbe. A D regiszterből az Y lapcímet átviszik az ELC első lapcím regiszterbe.

Ezután a következő szót alakítják ki a D regiszterben. Ez a szó tartalmazza a Z lapcímet, amely az LSZ lapszámlálóban levő lapszám, az USZSZ regiszterből vett SZSZ szegmensszámot, valamint a 0 betöltött állapotnak a kijelölését. Az új szó D(Y) részregisztere a „*" szegmensterminálló jelet kell, hogy tartalmazza, ha az USZLSZ regiszter elérte a 0-t. Mindegyik esetben az új szót ezt követően megfeleltetik—beírják az asszociatív tárba. Ha az USZLSZ új szegmens lapszám regiszter nem nulla, az LSZ lapszámlálót 1-gyel megnövelik és az argumentumot ismét előkészítik a következő kapcsolódó szónak egyezőszó-olvasására az asszociatív tárból, és egy másik új szónak egyezőszó helyére írására az asszociatív tárba. Ez a folyamat mindaddig folytatódik, amíg az USZLSZ regiszter 0 nem lesz (vagyis amíg az új szegmens utolsó lapját el nem érik). Ekkor befejeződik az allokáció első része.

Példaképpen allokáljunk egy 600 utasítás- és adatszóból álló szegmenst. Legyen 3 a hozzárendelt szegmensszám és a rendelkezésre álló szegmensnek (szegmensszám 9) a következő lapjai legyenek: 56—34—103—6—89—201—153—55. Az allokálás előtti asszociatív tár leképzést a 7.15*a* ábra mutatja. A szegmens első 128 szavát az 56. lapcímhez rendeljük; a hozzárendeléshez szükséges műveleteket a 7.15*a* ábra mutatja. A második 128 szó a 34. lapcímre kerül, és hasonlóképpen kerül kijelölésre a harmadik, negyedik és ötödik 128 szó; az ötödik lapon 40 szó kihasználatlan marad. Az asszociatív tár leképzést az allokáció után a 7.15*b* ábra mutatja.

A szegmens betöltésének második része most a kérdezőszó állapotának 0-ra változtatásával, valamint a kérdezőszó Y lapcímének, Z lapcímének és SZSZ szegmensszámának kimaszkolásával kezdődik. A KU(SC) részregiszterben levő sorcímet ellenőrzik, hogy meghatározzák, vajon elérte-e a 127-et. Ha nem 127, csak a sorcímet kell 1-gyel megnövelni, a lapcímnek nem kell változnia. Ha 127, új lapcímet kell hívni az asszociatív tárból, ez egy egyeztetési művelettel történik, ahol a KU(LC) részregiszterben levő lapcím a kérdező szó X lapcíme. A D tartalom- (puffer-) regiszterben levő Y lapcím az új lapcím az egyezőszó-olvasás művelet után; ezt a lapcímet most átviszik a KU(LC) részregiszterbe. Az egyezőszó-olvasás művelet után tesztelik mind a D(SZSZ), mind a D(Y) részregisztert. Ha a D(SZSZ) részregiszter nem egyezik a PSZSZ regiszterben levő szegmensszámmal, megszakítás kezdődik. Ez az összehasonlítás tárvédelem céljára szolgál. Ha a D(Y) részregiszter a „*" szegmensterminálló szimbólumot tartalmazza, a szegmens betöltése befejeződik.

Függetlenül attól, hogy a KU(SC) részregiszter a 127-et tartalmazza-e vagy sem, a KU regiszterben levő tárcím most átvitelre kerül a CR címregiszterbe és a TSIN társínnél készenlétben levő főtárszót tárolják a főtárban. A betöltő most visszatér és ismét ellenőrzi a KU(SC) részregiszterben levő sorcímet. Ez a folyamat addig folytatódik, amíg el nem éri a szegmensterminálló szimbólumot.

A 7.14. folyamatábrán látható allokációs és betöltési folyamatokat a következő procedurális utasítások írják le.

```
Comment, dinamikus betöltő allokációs művelet sorozata                (7.24)
Comment, kezdetiérték-adás
      LSZ←0, A(X,Y,Z,SZSZ,S)←ELC—0—0—9—4, M(X,Y,Z,SZSZ,S)←
        ←255—0—0—31—7;
      KU(LC,SC)←ELC—0, PSZSZ←USZSZ;

Comment, itt kezdődik az allokációs hurok
/W/   USZLSZ←countdn USZLSZ, EO←1,;
      ELC←D(Y), D(Z,SZSZ,S)←LSZ—USZSZ—0;
      IF (USZLSZ=0) THEN (GOTO X) ELSE (EI←1);
      LSZ←countup LSZ, A(X)←ELC, GOTO W;
/X/   D(Y)←*;
      EO←1;
```

X lapcím	Y lapcím	Z lapcím	Szegmens-szám	Állapot
6	89		9	100
34	103		9	100
55	*		9	100
56	34		9	100
89	201		9	100
103	6		9	100
153	55		9	100
201	153		9	100
56			3	110
			9	110

0
3
000
Szegmens-szám-regiszterek
1
8 ELC
0 LSZ

a)

X lapcím	Y lapcím	Z lapcím	Szegmens-szám	Állapot
56	34	0	3	000
34	103	1	3	000
103	6	2	3	000
6	89	3	3	000
89	*	4	3	000
201	153		9	100
153	55		9	100
55	*		9	100
56			3	110
201			9	110

Betöltött szegmensek
Rendelkezésre álló szegmens
Szegmens-szám-regiszterek
201 ELC
4 LSZ

b)

7.15. ábra
Példa asszociatívtár-térképre
***a)* allokálás előtt; *b)* allokálás után**

```
Comment, dinamikus betöltő betöltési sorozata
Comment, kezdetiérték-adás
        A(S)←0, M(X,Y,Z,SZSZ,S)←255—0—0—0—7;
Comment, itt kezdődik a betöltési hurok
/Y/     IF(KU(SC)=127) THEN (GOTO Z);
        A(X)←KU(LC);
        MEO←1;
        IF (D(SZSZ)≠PSZSZ) THEN (GOTO MEGSZAKÍTÁS);
        IF (D(Y)=*) THEN (GOTO VISSZA) ELSE (KU(LC)←D(Y));
/Z/     KU(SC)←countup KU(SC);
        CR←KU;
        TR←TSIN;
        FT(CR)←TR, GOTO Y;
        END
```

7.5.3. Utasítássorrend

Az utasítássorrendet a KU következő utasítás regiszter vezérli; a legkisebb helyiértékű 7 bitje a sorcímszámláló a sorcímek növekvő sorrendjének biztosításához, a legnagyobb helyiértékű a 8 bit pedig a lapcímet tárolja. Egy szegmens végrehajtása a szegmens első lapjának első sorával kezdődik. A lapon belül a sorcímet a sorcímszámláló tartalmazza. Amikor a sorcímszámláló új ciklust kezd a 127 értékről a 0-ra, új lapot kell kivenni az asszociatív tárból. Az a mód, ahogyan az új lapot és sorcímet előállítják, azonos azzal, amit a betöltési folyamat második részében leírtunk. Vegyük figyelembe, hogy a szegmens utasításainak aktuális tárbeli címei mindaddig nem ismeretesek, amíg ezek az utasítások végrehajtásra nem kerülnek.

7.5.4. Operanduscím lehívása

A végrehajtásra kerülő utasításnak 15-bites operanduscíme van. Ez a cím a szegmens kezdőpontjához van viszonyítva, de az aktuális tárbeli címhez nem, mivel a szegmens utasításainak operanduscímei nem változtak meg a betöltés folyamán. Ezért meg kell találni azt az aktuális operanduscímet, amelyen a főtár tárolja az operandust. Ennek az aktuális operanduscímnek a sorcíme viszont ugyanaz, mint az eredeti legkisebb helyiértékű 7-bites cím, amely most éppen a TR tartalom- (puffer-) regiszterben van és a CR(SC) részregiszterbe vittek át. Az aktuális operanduscím lapcíme az asszociatív tárból kereshető ki, az argumentumban levő Z szegmensen belüli lapcím felhasználásával (Z-t vesszük az X lapcím helyett, mint az utasítás sorrendbe állítása esetén). Az egyezőszó-olvasás művelet után, a D tartalom- (puffer-) regiszterben levő X alap cím az aktuális operanduscím lapcíme, amit azután a CR(LC) részregiszterbe visznek át, az operandus hívása céljából.

7.5.5. Elágazás, indexelés és indirekt címzés

Elágazás jöhet létre, amikor átviteli utasítást hajtanak végre. Ebben az esetben az operanduscím egy utasításcím. Az operanduscímet ismét lehívják, ugyanúgy, ahogy az előzőkben leírtuk, azzal a különbséggel, hogy most a TR tartalom- (puffer-) regiszterben levő sorcímet és az asszociatív tárból a lapcímet viszik át a következő utasítás címregiszterbe, a CR címregiszterben levő eredeti cím helyett.

A fenti operanduscím-lehívás nem gátolja meg az indexelés és indirekt címzés használatát. Ahhoz, hogy járulékosan indexelni is lehessen, az indexregiszter tartalmát hozzá kell adni (vagy le kell vonni) az eredeti relatív címhez (vagy címből), mielőtt az operanduscímet lehívnák. Indirekt címzéshez operanduscímet kell egy operandus helyett hívni, és az operandushívást meg kell ismételni.

7.5.6. Programvisszatérés és tárfelszabadítás

Ha megszakítás történik egy szegmens utasításainak végrehajtása során, a felügyelőprogram a visszatérési címet (mind a lap-, mind a sorcímet) az ehhez a szegmenshez hozzárendelt szegmensszámregiszterben helyezi el (ezeket a 7.13. ábrán P1-, P2-, P3- és P4-gyel jelöltük), és ezután másik szegmenst választ ki végrehajtásra. Mindannyiszor, valahányszor újrakezdődik az eredeti szegmens végrehajtása, a felügyelőprogram megkapja a visszatérési címet az eredeti szegmenshez rendelt szegmensszámregiszterből.

Amikor egy szegmens végrehajtása befejeződött, a felügyelőprogram dönthet arról, hogy felszabadítja-e a szegmens lapjait. A felszabadítás egyszerűen a lapoknak a rendelkezésre álló szegmenshez való hozzákapcsolásával történhet.

7.6. Puffertár

A viszonylag gyors, de kisméretű főtárból, — amilyen pl. a mágnesmagos tár — és a viszonylag lassú, de nagyméretű háttértárakból — mint pl. a mágnesdob vagy mágneslemez — alkotott tárhierarchiát azóta alkalmazzák, amióta először építettek nagyméretű digitális számítógépet. Az ilyen tárhierarchia kialakításakor az a cél, hogy a háttértár szolgáltassa a szükséges nagy tárkapacitást, a főtár pedig a kívánt feldolgozási sebességet. Ilyen tárhierarchia μs...ms tartományban működik.

Az elmúlt években fontos fejlődést jelentett a társzervezésben a tárak elérési idejének ns...μs tartományba történő helyezése. Ezt igyekezett megvalósítani — bár sokszor egyszerű formában — számos számítógép [27], [29], [30] regiszterek, sőt igen kis kapacitású tárak alkalmazásával is. Erre vonatkozóan volt javaslata 1962-ben Blommnak [28] és 1963-ban Leenek [31] „look-aside-memory" (oldalttekintő memória) elnevezéssel, 1965-ben Wilkes-nek [33], „slave-memory" elnevezéssel, amelyet segédtárnak néha, szószerinti fordításban „szolga"-tárolónak nevezhetünk. Elsőként

puffertárat az IBM System/360 85-ös modelljében valósítottak meg [37], amelynek pufferjét cache-nek hívják és a programozó számára „átlátszó" (nem kell figyelembe vennie).

Ebben a fejezetben a puffertárak szervezését és az IBM System/360 85-ös modelljében megvalósítotthoz hasonló tárak szervezését írjuk le.

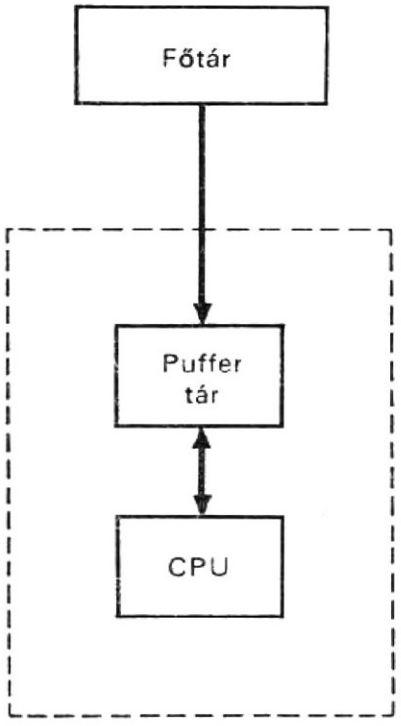
7.6.1. Puffertár szervezése

Általános esetben egy számítógéprendszer főtárához a központi egység (CPU) tud fordulni egyszerre egy tárbeli szóért. A CPU feldolgozási idejét a főtár sebessége (ciklusideje) korlátozza. A ciklusidőből adódó korlátozás egyre kritikusabbá válik, ahogyan a főtár kapacitása mind nagyobb és nagyobb lesz, és a CPU sebessége mind gyorsabban növekszik.

Ha a főtárhoz pufferként egy nagyságrenddel gyorsabb, kis kapacitású tárat használnak (7.16. ábra), a CPU sebessége nagymértékben megnő, mivel a főtárhivatkozásokat erőteljesen csökkenteni lehet. Ennek okai:

1. Az átvitel a főtárból a puffertárba blokkonként történhet, azaz több szóval lehet egyidejűleg átvitelt végezni. Ha a főtár többutas beékelésű (interleaving), a szavakból álló blokkot egyetlen főtár-ciklusidőben lehet átvinni.
2. A blokkátvitel a kívánt szavakat előre behívhatja a puffertárba és a CPU rendelkezésére bocsáthatja, mivel elég nagy a valószínűsége annak, hogy a hivatkozott blokk többi szavára lesz rövidesen szükség.
3. A puffertárban levő szavak többször is felhasználásra kerülhetnek a programban levő iteratív hurkok és szubrutinok miatt, így nagymértékben csökken a főtárhoz fordulások száma.

A következő leíráshoz a 7.5. táblázatban feltüntetett jellemzőjű főtárat és puffertárat választottuk. A főtár ciklusideje 1 μs, adatátviteli út szélessége (azaz szóhossza) 128 bit, kapacitása pedig 64K szó, 128-bites szavakból (ahol K az 1024-es szorzót jelenti), továbbá négyutasan van beékelve. A puffertár ciklusideje 80 ns, az adatátviteli



7.16. ábra.
Puffertár szervezése

7.5. táblázat

A főtár és a puffertár adatai

Jellemző	Főtár	Puffertár
Tárciklusidő	1 μs	0,08 μs*
Adatátviteliút szélessége	128 bit vagy 1 szó	128 bit vagy 1 szó
Adategység	*a)* 128 bit/szó	*a)* 128 bit/szó
	b) 4 szó/blokk	*b)* 4 szó/blokk
	c) 16 blokk/lap	*c)* 16 blokk/lap
Tárkapacitás**	*a)* 64K szó	*a)* 1024 szó
	b) 16K blokk vagy	*b)* 256 blokk vagy
	c) 1K lap	*c)* 16 lap
Beékelés	Négyutas	Nincs
Címregiszter	16 bit	10 bit

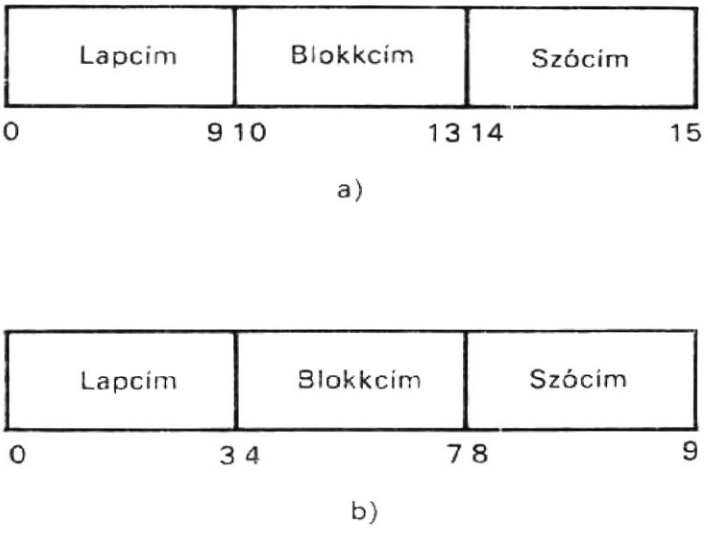
* A CPU ciklusideje is 0,08 μs.
** A K 1024-gyel való szorzást jelent.

út szélessége szintén 128 bit és a kapacitása 1024 szó. Mindkét tár négyszavas blokkokra van osztva; tehát 16K blokk van a főtárban és 256 blokk a puffertárban. Minden 16 folytonos blokk egy lapot alkot; tehát 1K lap van a főtárban és 16 lap a puffertárban. A főtár és a puffertár között egy blokk, a puffertár és a CPU között pedig egy szó kerül egyszerre átvitelre. A főtár 16-bites, a puffertár pedig 10-bites című. Formátumaik a 7.17. ábrán láthatók. A főtárcím egy 10-bites lapcímből, 4-bites blokkcímből és 2-bites szócímből áll. A puffertár címregiszterének formátuma azonos az előzőével, azzal az eltéréssel, hogy a lapcím 4-bites.

7.6.2. Puffertár szervezése

Az itt leírásra kerülő puffertár szervezéshez feltételezzük, hogy a főtár első lapja nem használható; tehát a főtár 0-ás lapcíme nem fordulhat elő.

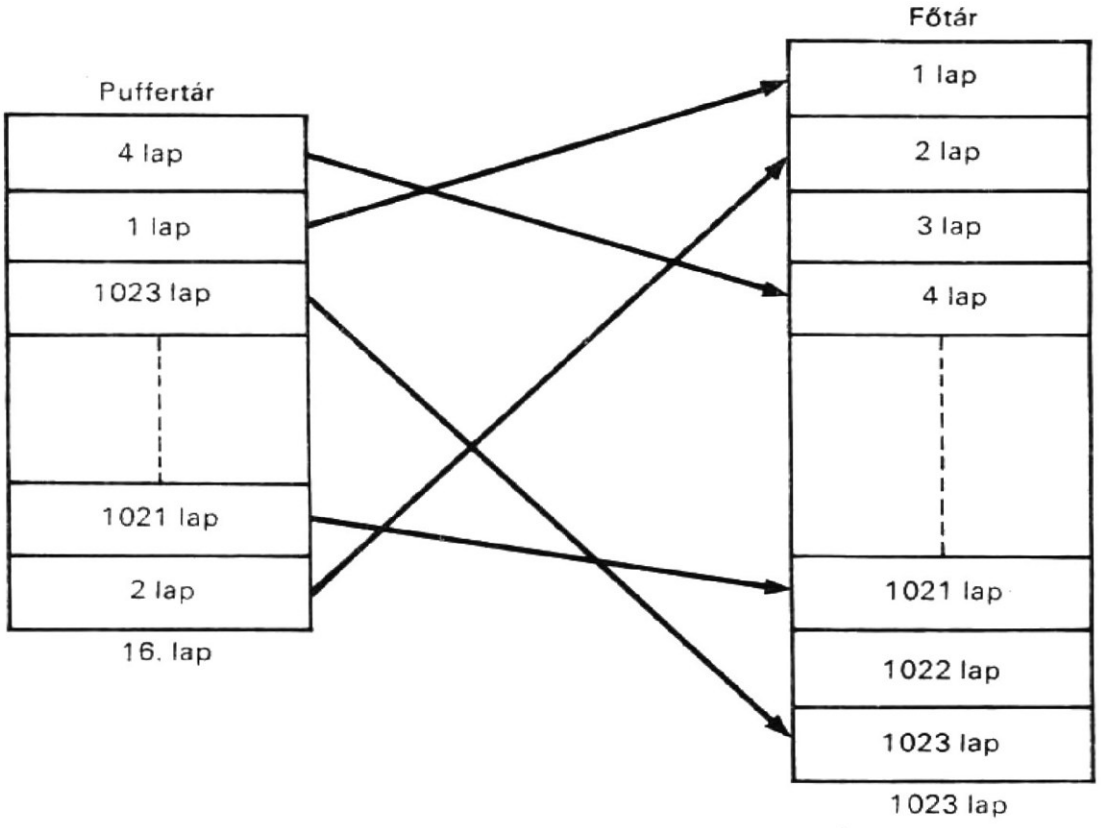
Mint említettük, mind a főtár, mind a puffertár lapokra van osztva. Működése közben a főtár 1023 lapja közül 16 a puffertár 16 lapján van tárolva. Ezt a 16 lapot a főtárban elfoglalt lapcímük jelöli a 16 lapcímregiszterből álló tömbben. A lapleképezésnek és a lapcímmegjelölésnek ezt az elrendezését a 7.18. ábrán láthatjuk.



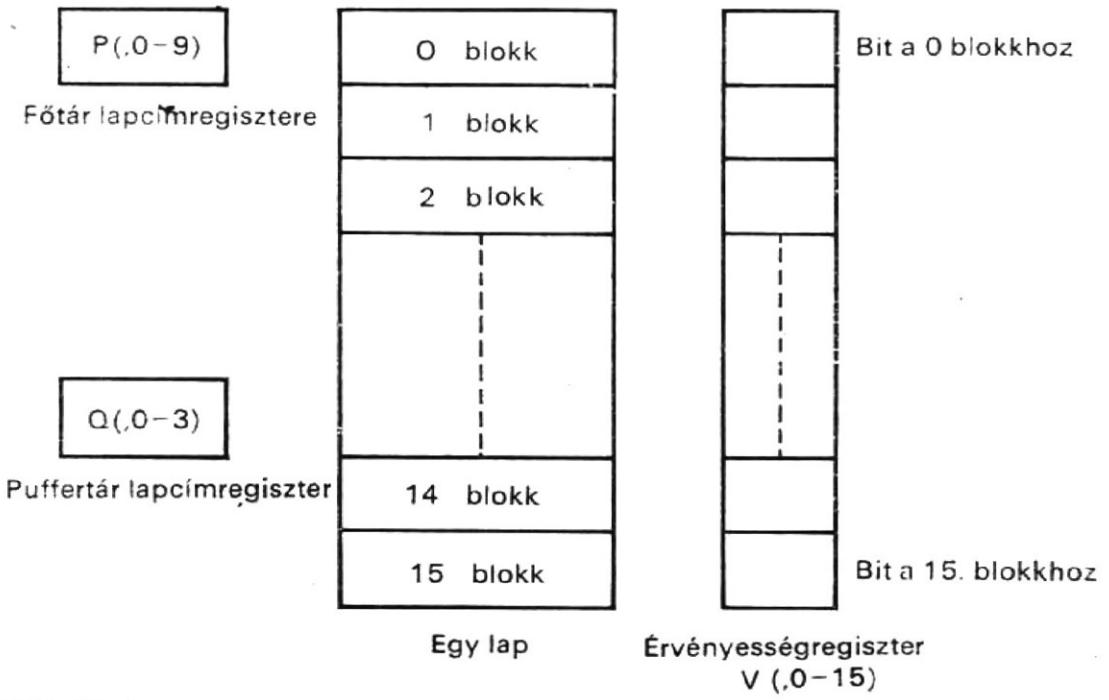
7.17. ábra
Tárak címformátuma
a) **főtár címformátuma;** *b)* **puffertár címformátuma**

A főtárnak és a puffertárnak az előbbiek szerint tehát valamennyi lapja 16 blokkra van osztva. A puffertár egy lapján levő 16 blokkot ábrázolja a 7.19. ábra. Ezek a blokkok négy szóra vannak felosztva (ez már az ábrán nem látható). Amint a 7.19. ábra is mutatja, a puffertár minden lapjához tartozik:

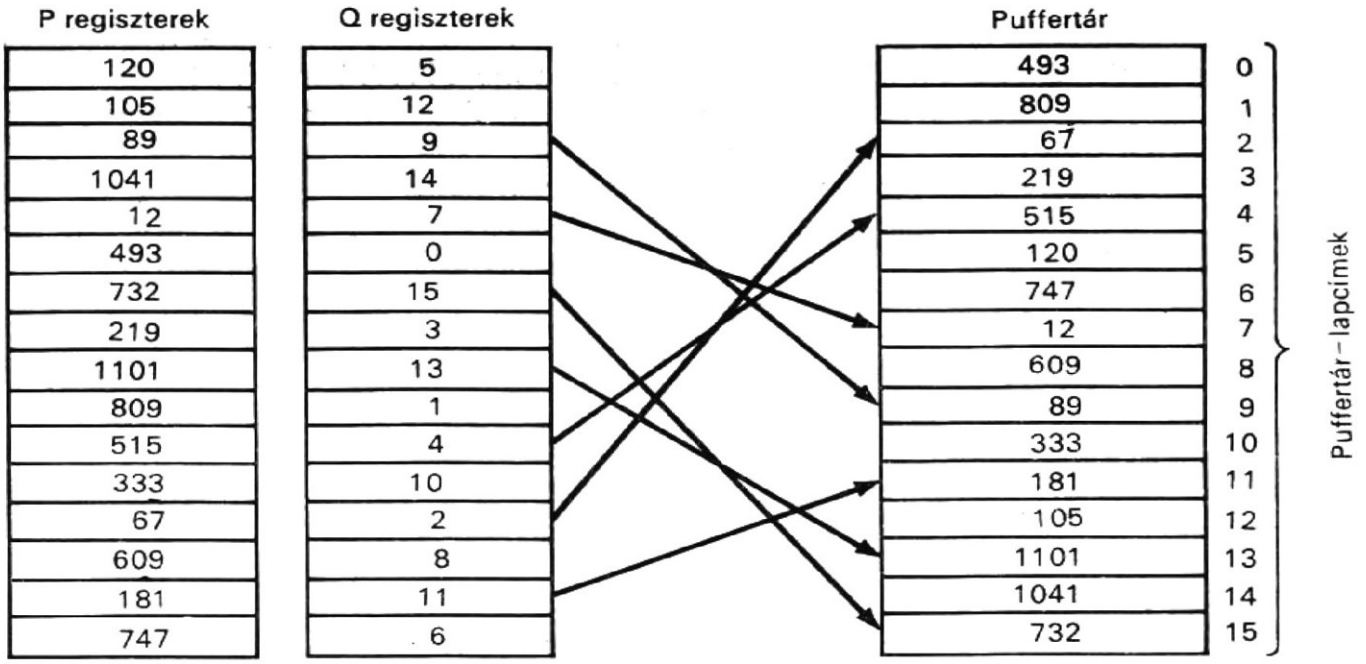
— egy regiszter, amely a 10-bites főtárlapcímet tartalmazza;
— egy 16-bites blokk betöltött állapotát jelző regiszter, amelynek 16 bitje tárolja a lap 16 blokkjának állapotát (az 1 jelentése: a blokk a puffertárban van) és
— egy további regiszter, amelyben a puffertár 4-bites lapcíme helyezkedik el.



7.18. ábra
A főtárban és a puffertárban levő lapok közötti leképezés



7.19. ábra
Blokkok egy lapon és a hozzájuk tartozó P, Q és V regiszterek (16 blokk van egy lapon és 4 szó van egy blokkban)



7.20. ábra
Fordítás a főtár lapcímei és a puffertár lapcímei között, a P és Q regisztertömbök felhasználásával

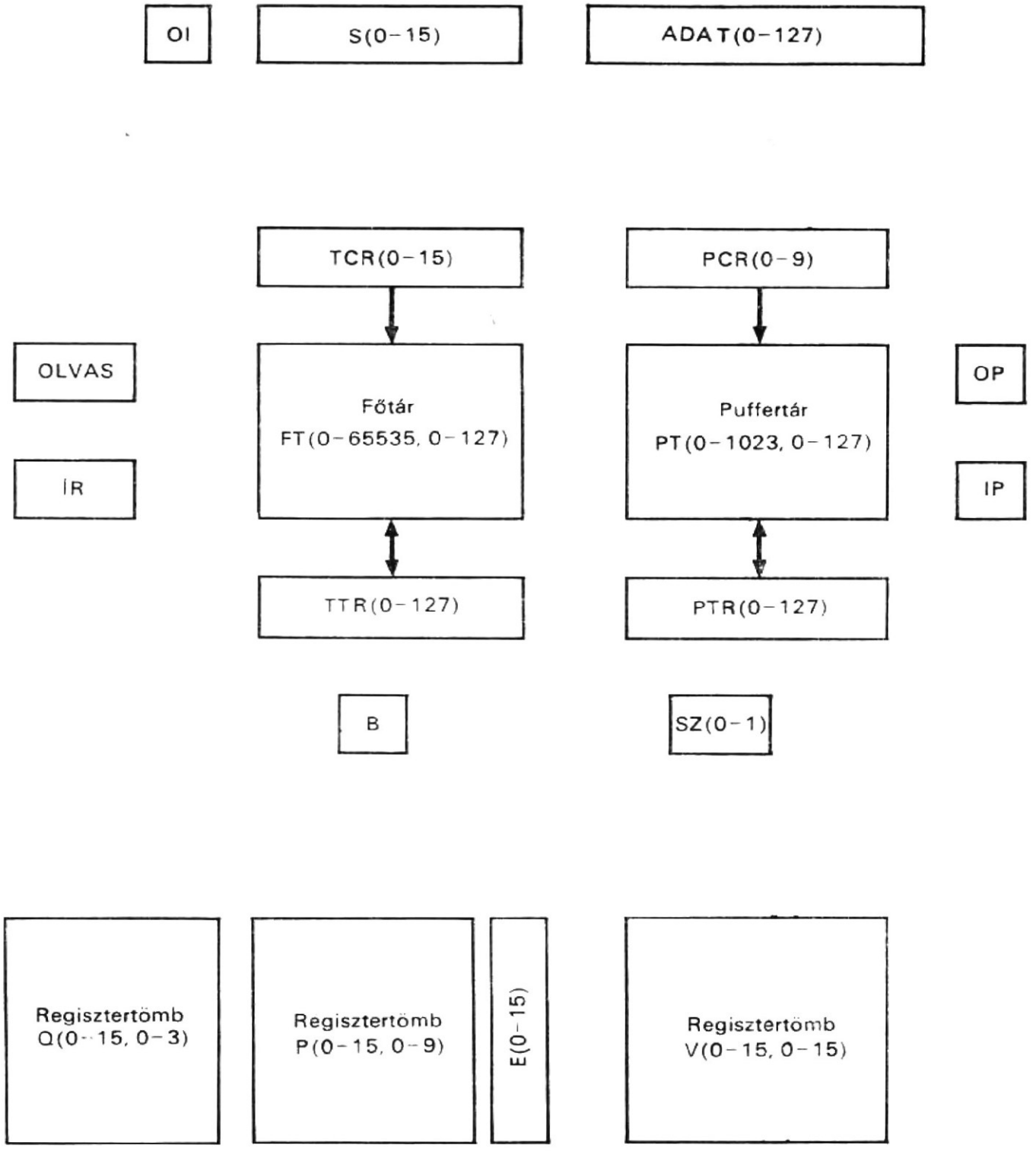
Minthogy 16 lap van a puffertárban, ezért 16 főtár-lapcímregiszterből álló P tömböt, 16 érvényességregiszterből álló V tömböt és 16 puffertár-lapcímregiszterből álló Q tömböt kell alkalmazni. Tehát a puffertár egy lapjához egy a blokk betöltött állapotát jelző regiszter, egy főtár-lapcímregiszter és egy puffertár-lapcímregiszter tartozik.

Mint az előzőekben említettük, a puffertár minden lapjához egy-egy, a P és Q regiszterekből álló páros tartozik. A P regiszter tárolja a lap főtárlapcímét a puffertárban; a Q regiszter tárolja a puffertárnak azt a lapcímét, ahol ezt a lapot a puffertár tárolja. Ezt illusztrálja a 7.20. ábra. Látható, hogy a puffertár számai a főtár lapcímei; ezek éppen azok a lapok, amelyeket e lapcímek címeznek. Továbbá, amint a későbbiekben leírjuk, előfordulhat, hogy ezek közül nem mindegyiket tárolták a puffertárban.

A lapcímregiszterek P tömbjének három funkciója van. Az első funkció, amint említettük, a főtárban levő azon 16 lap lapcímének tárolása, amelyek (részben vagy teljesen) a puffertárban vannak. A P tömb második funkciója szerint asszociatív tárként működik, és így egy lapcím megadásakor szimultán összehasonlítások történnek a P tömbben levő címekkel; az egyeztetett pozíciókat a hozzárendelt E egyezést jelző regiszter jelzi. A harmadik funkció egy *aktivitáslista* tárolása. Itt a CPU által legutoljára hivatkozott lapcímet helyezik el a lista tetjén, míg azt a lapcímet, amelynek lapját a legközelebbi alkalommal helyettesíteni kell a puffertárban, a lista aljára teszik.

A fentiekben leírt puffertár — főtár struktúra a 7.21. ábrán levő blokkdiagramon látható. Az FT főtárhoz tartozik a TCR címregiszter, a TTR tartalom- (puffer-) regiszter, valamint az OF és IF olvasás és írás vezérlőregiszter. A PT puffertárhoz tartozik a PCR címregiszter, a PTR tartalom- (puffer-) regiszter, valamint az OP és IP olvasás és írás vezérlőregiszter. A CPU által rendelkezésre bocsátott tényleges címet, az adatszót és az olvasás —[írás parancsot sorra az S, ADAT és az OI regiszterek tárolják. Végül az SZ jelű regiszter mint számláló működik, a B regisztert pedig a pufferelérési műveletsorozat vezérlésére használják, amint azt a későbbiekben bemutatjuk. Ezt a struktúrát a következő CDL utasítások írják le:

```
Comment, puffertár — főtár struktúrája                              (7.25)
Comment, puffertár vezérlőregiszterei

Array-register, P(0—15, 0—9),      $főtár-lapcím-regisztertömbje
                Q(0—15, 0—3),      $puffertár-lapcímregisztertömbje
                V(0—15, 0—15),     $blokkérvényesség-regisztertömb
```



7.21. ábra
Főtár—puffertár szervezése és struktúrája

```
Register,       E(0—15),          $egyezést jelző regiszter a P regisztertömbhöz
                SZ(0—1),          $számláló
                B,                $pufferelérési műveletsorozat vezérlőregisz-
                                  tere
Encoder,        N(0—3)=E,
Comment, CPU regiszterei
Register,       S(0—15),          $CPU tényleges címregiszter
                ADAT(0—127),      $CPU tartalom- (adat-) regiszter
                OI,               $CPU olvasás—írás parancs regiszter
Subregister,    S(LC,BC,SZC)=S(0—9,10—13,14—15),
```

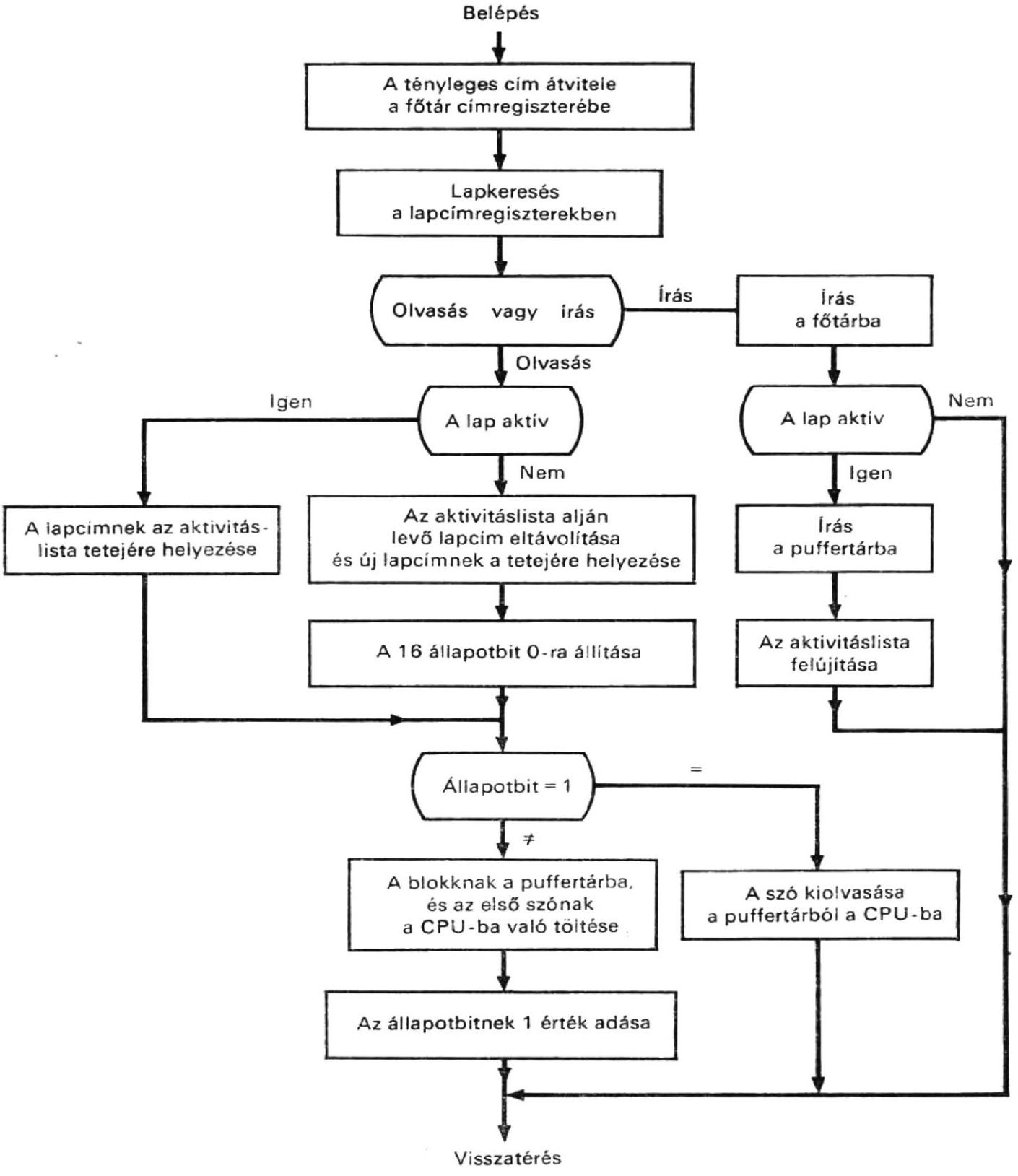
```
Comment, fő- és puffertár és a hozzájuk tartozó regiszterek
Register,          TCR(0—15),          $főtár-címregiszter
                   TTR(0—127),         $főtár-tartalomregiszter
                   OF,                 $főtár olvasásparancs
                   IF,                 $főtár írásparancs
                   PCR(0—9),           $puffertár-címregiszter
                   PTR(0—127),         $puffertár-tartalomregiszter
                   OP,                 $puffertár olvasásparancs
                   IP,                 $puffertár írásparancs
Memory,            FT(TCR)=FT(0—65535, 0—127),
                   PT(PCR)=PT(0—1023, 0—127),
Block, FELÚJÍT     (IF(E(1)=1 THEN (P(0—1,)←cir P(0—1,)
                                   (Q(0—1,)←cir Q(0—1,),
                                   (V(0—1,)←cir V(0—1,)),
                   IF (E(2)=1) THEN (P(0—2)←cir P(0—2,),Q(0—2,)←cir
                                   Q(0—2,), V(0—2,)←cir V(0—2,)),
                   .....................................................
                   IF (E(15)=1) THEN (P(0—15,)←cir P(0—15,), Q(0—15,)←cir
                                   Q(0—15,), V(0—15,)←cir V(0—15,)),
Operator,          J(0—15)←K(0—15,) megfelel L
Register,          L(0—9), J(0—15),
Array-register,    K(0—15, 0—9),
/begin/            J(0)←(K(0,0) ⊙L(0)) * (K(0,1) ⊙L(1)) *
                   * ..... * (K(0,9) ⊙L(9)),
                   J(1)←(K(1,0) ⊙L(0)) * (K(1,1) ⊙L(1)) *
                   * ..... * (K(1,9) ⊙L(9)),
                   ..................................
                   J(15)←(K(15,0) ⊙L(0)) * (K(15,1) ⊙L(1)) *
                   * ..... * (K(15,9) ⊙L(9)),
                   end of operator
```

A fenti kódoló az E egyezést jelző regiszter tartalmát puffertár-lapcímmé kódolja. A FELÚJÍT mikroműveletek az aktivitáslistát frissítik fel, amint azt a későbbiekben le. fogjuk írni. A „megfelel" operátort úgy definiáltuk, mint az adott főtárlapcím és a P regiszterekben levő 16 cím közötti megfelelést. Ezt a definíciót el lehetett volna kerülni, ha a korábbiakban már ismertetett operátort használjuk.

A puffertár egy szó eléréséhez tartozó műveletsorozatát a 7.22. folyamatábrán láthatjuk. Amikor a CPU tárhivatkozást kér, a tényleges címet átírják a főtár címregiszterébe. Ezután végigkeresik a P regisztertömböt a tényleges lapcímért.

Olvasási művelethez — ha a lap aktív — a lapcímet az aktivitáslista tetejére helyezik. Ha a lap nem aktív, eltávolítják a lapcímet az aktivitáslista aljáról. Az új lapcímet a lista tetejére helyezik és a hozzá tartozó blokk betöltött állapotát jelző regisztert visszaállítják 0-ra, hogy jelezze: a puffertárban levő 16 blokk közül egyiket sem most töltötték be a főtárból. Mindegyik esetben megvizsgálják a tényleges blokkcímhez tartozó állapotbitet. Ha az állapotbit 1, a blokk a puffertárban van, és ekkor a szót kiolvassák a puffertárból. Ha az állapotbit 0, legközelebb a blokkot és (nem a lapot) töltik be a főtárból a puffertárba. A betöltés alatt a főtárból az első szót a CPU-ba is átviszik.

Írásnál a szót mindig a főtárba írják be; ezt storage through (tárolón keresztül) műveletnek nevezik. Ha a lap aktív, a szót a puffertárba is beírják, és az aktivitáslistát felfrissítik. A kettős felfrissítésre azért van szükség, mert a be-, ill. kimeneti csatornák szintén kapcsolatot tartanak a főtárral.



7.22. ábra
Puffertár olvasási és írási műveleteinek hatásvázlata

Megemlítjük, hogy a csatornák a CPU-n keresztül, de közvetlenül a főtárból hívnak adatokat anélkül, hogy átmennének a puffertáron. A csatornatárak kezelése ugyanolyan, mint a CPU-táraké; tehát, ha egy csatorna változtat adatokat a puffertárban, a puffert frissítik fel.

7.6.3.
Puffertár elérési műveletsorozata

A puffertár elérésére szolgáló sorrendi műveleteket *pufferelérési sorozatnak* nevezik. Ezt a sorozatot a B regiszter vezérli. A pufferelérési műveletsorozat a 7.23. folyamatábrán látható. Feltételezzük, hogy a CPU kezdetben a tényleges címet, az adatszót (ha van ilyen) és az olvasás — írás parancsot az S, a ADAT ill. az OI regiszterekben helyezi el, Emlékezzünk vissza, hogy kiindulásunkkor feltételeztük, hogy a főtár 0 lapcíme nem fordulhat elő.

Amint a 7.23. ábrán látható, a pufferelérési művelet sorozat akkor aktivizálódik, amikor a B regiszter 1 értéket kap. Az S regiszterben levő tényleges címet átviszik a TCR főtár-címregiszterbe. Ezután P regisztereket egyeztetésvizsgálattal megnézik, hogy nincs-e olyan lapcíme, amely megegyezik az S(LC) részregiszterben levő tényleges lapcímmel. Azokat a pozíciókat, amelyek megegyeznek, megjelölik az E egyezést jelző regiszterben. Ettől a ponttól kezdve, a további műveletek attól függnek, hogy a CPU olvasási vagy írási parancsot adott-e ki.

Ha a parancs írás, az ADAT regiszterben levő adatszót beírják a főtárba és, ha az E egyezést jelző regiszter nem tartalmaz 0-t, felfrissítik az aktivitáslistát az egyeztetett lapcímnek a lista tetejére helyezésével (amint azt a továbbiakban leírjuk), és az adatszót beírják a puffertárba. Ezután visszaállítják 0-ra a B és E regisztereket. A műveletsorozat ezzel be is fejeződött.

Ha a CPU olvasási parancsot ad, megvizsgálják az E egyezést jelző regisztert, hogy meghatározzák, aktív-e a lap. Ha a lap nem aktív, mind a P, mind a V regisztertömböket jobbra léptetik, a Q regisztertömböt pedig ciklikusan léptetik jobbra. Ez a módszer, amellyel a P regisztereket léptetik, a tényleges lapcímet az aktivitáslista tetejére helyezi és eltávolítja a lapcímet a lista aljáról (7.24*a* ábra). A Q regiszterek tartalmának léptetésével az újonnan rendelkezésre álló puffertár lapcíme a most az aktivitáslista tetején elhelyezkedő effektív lapcímhez fűződik hozzá. A V regiszterek tartalmának léptetésével 0-ra állnak vissza azok a blokkállapotbitek, amelyek a tényleges lapcímhez tartoznak. Ha a lap aktív, a (7.25) utasítások blokkutasításával definiált FELÚJÍT mikroműveletek kerülnek végrehajtásra. Ezek a mikroműveletek az egyeztetett lapcímet az aktivitáslista tetejére helyezik, és a közbenső lapcímeket egy pozícióval lejjebb viszik (7.24*b* ábra). Ezek egyúttal azok a mikroműveletek, amelyek az aktivitáslista felfrissítéséhez szükségesek a beírási művelet folyamán, ha az E regiszter nem tartalmaz 0-t. Miközben a P regiszterekben levő lapcímeket mozgatják, hasonlóképpen mozgatják a V regiszterekben levő állapotbiteket és a Q regiszterben elhelyezkedő puffertár-lapcímeket. Az aktivitáslista mozgatásával (l. 7.24. ábra) a legkevésbé aktív lapcím a lista aljának irányába tolódik, sőt, esetleg meg is szűnik, ha a lapcímre már hosszabb ideje nem történt hivatkozás.

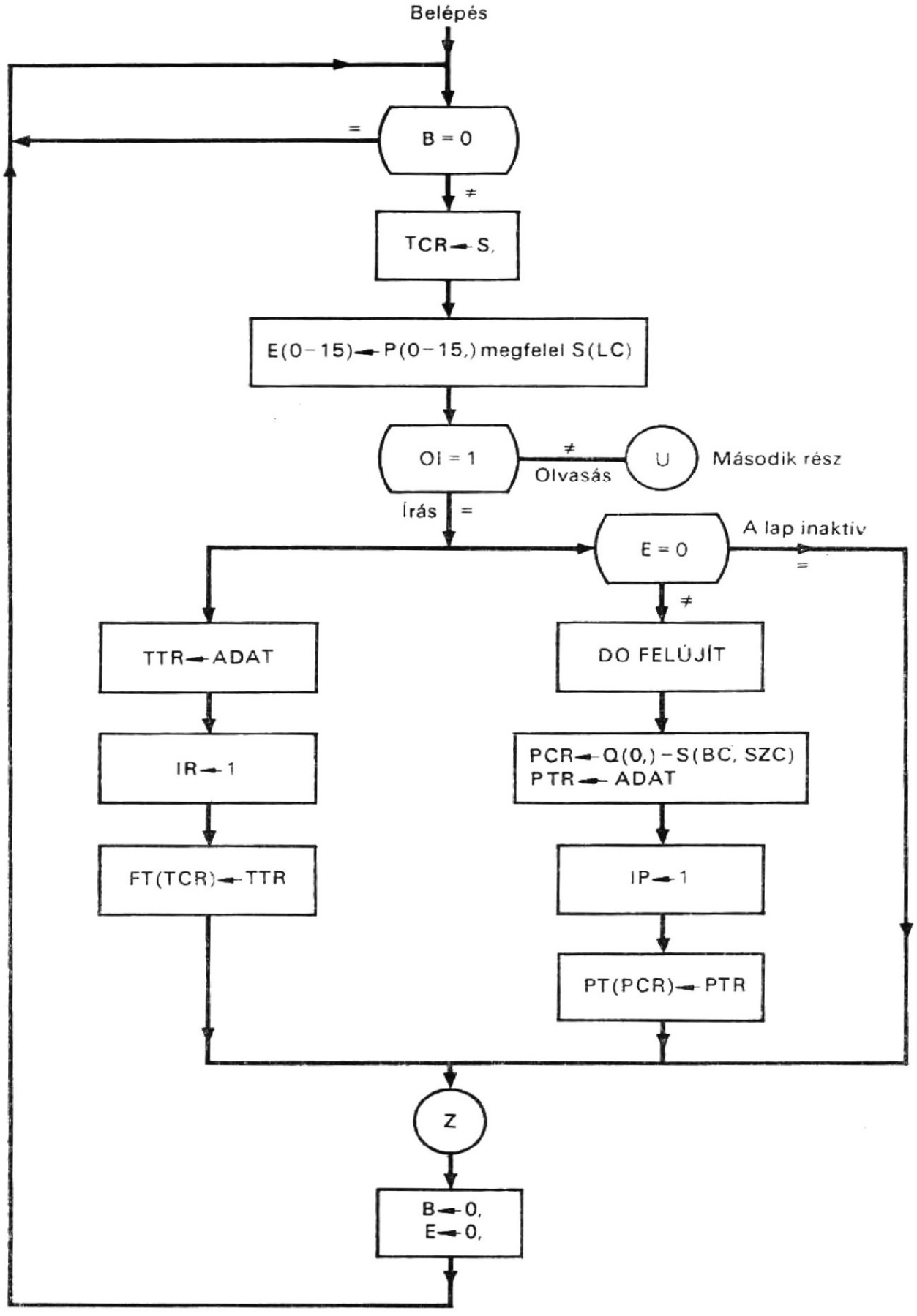
Miután az aktivitáslistát egy olvasási művelet kérés eredményeképpen felfrissítették, megvizsgálják az S(BC) részregiszterben levő blokkcím által specifikált állapotbitet. Minthogy a pontos állapotbit egy meghatározott blokkcímtől függ, ezt az állapotbitet a következő szimbolikus indexszel címezik meg:

$$V(0,S(BC)), \tag{7.26}$$

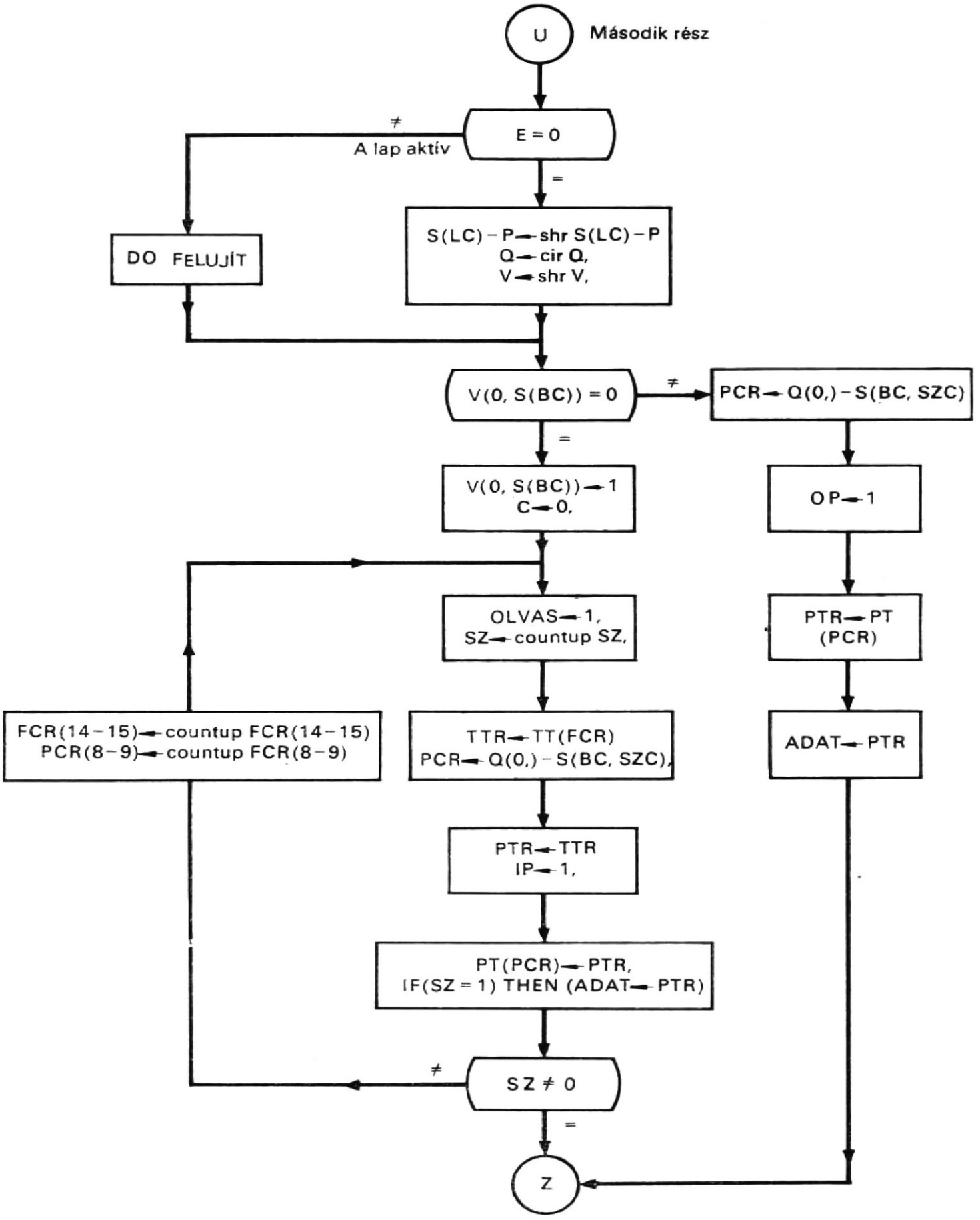
Ennek a speciális érvényességi bitnek az ismertetett címzési módszere a következő 16 feltételes mikroutasítás leírásával azonos:

IF (S(BC)=0) THEN (V(0,0).....),
IF (S(BC)=1) THEN (V(0,1) .....),

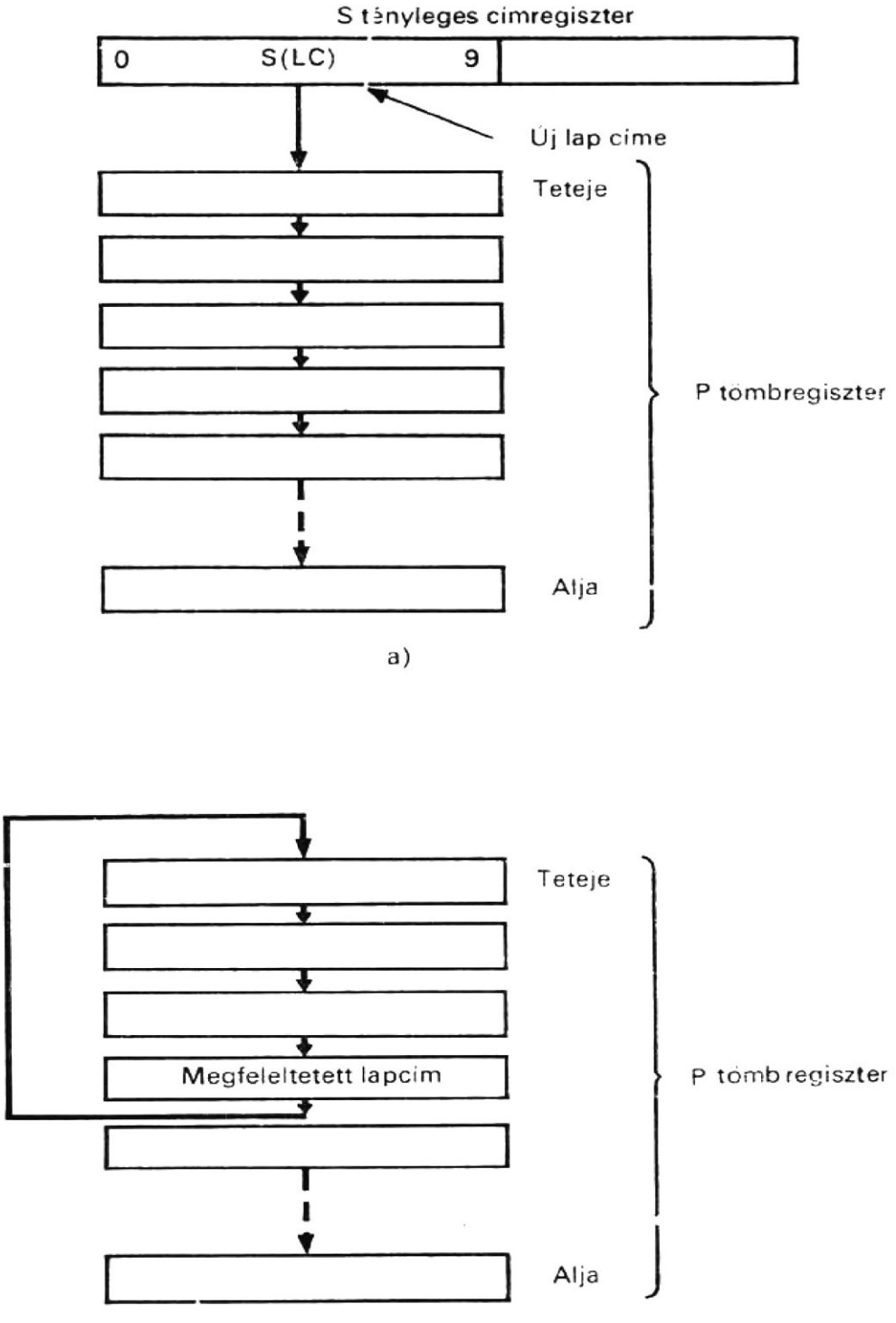
.....
IF (S(BC)=15) THEN (V(0,15) .....),



7.23. ábra
Folyamatábra puffertár elérési műveletsorozatának szemléltetésére


U
Második rész
E = 0
≠
A lap aktív
=
DO FELUJÍT
S(LC) – P ← shr S(LC) – P
Q ← cir Q,
V ← shr V,
V(0, S(BC)) = 0
≠
PCR ← Q(0,) – S(BC, SZC)
=
V(0, S(BC)) ← 1
C ← 0,
OP ← 1
OLVAS ← 1,
SZ ← countup SZ,
PTR ← PT
(PCR)
FCR(14–15) ← countup FCR(14–15)
PCR(8–9) ← countup FCR(8–9)
TTR ← TT(FCR)
PCR ← Q(0,) – S(BC, SZC),
ADAT ← PTR
PTR ← TTR
IP ← 1,
PT(PCR) ← PTR,
IF(SZ = 1) THEN (ADAT ← PTR)
≠
SZ ≠ 0
=
Z

7.23/b ábra



7.24. ábra
Az aktivitáslista felfrissítésének két útja
***a)* az új lap címének az aktivitáslista tetejére helyezésével;**
***b)* az egyeztetett lapcímnek az aktivitáslista tetejére helyezésével**

Ez a leírás túlságosan is hosszadalmas ahhoz, hogy könnyen használható legyen; ezért a (7.26) szimbolikus formát fogadjuk el. Ha az állapotbit 1, a szóblokk a puffertárban van, és a meghatározott szót a puffertárból kiolvasva az ADAT regiszterbe kerül. A puffertárcímet úgy alakítják ki, hogya Q(0,) regiszter tartalmát lapcímnek és az S(BC,SZC) részregiszter tartalmát blokkcímnek és szócímnek használják. Ha az állapotbit 0, a szóblokk nincs a puffertárban; ezt a szóblokkot most betöltik a tárba és az állapotbitnek 1 értéket adnak. A betöltés alatt az SZ számlálót használják a szavak számának számlálására. Az első szót a főtárból a CPU-ba is átviszik, így csökkenthető az elérési idő. A B és E regisztereket ezután visszaállítják 0-ra. Ezzel a művelet sorozat befejeződött.

A 7.23 ábrán látható pufferelérési művelet sorozatot a következő procedurális utasításokkal lehet leírni:

```
Comment, itt kezdődik a puffertárelérési sorozat                    (7.27)
(W)      IF(B=0) THEN (GOTO W);

Comment, a tényleges cím átvitele a címregiszterekbe
                TCR←S;

Comment, lapkeresés a P tömbregiszterben E(0—15)←P(0—15), megfelel S(LC);
Comment, olvasás vagy írás meghatározása
         IF (OI=I) THEN (GOTO Y);

Comment, mikroműveletek az olvasásparancshoz
(U)      IF (E=O) THEN (S(LC)—P←shr S(LC)—P,
         Q←cir, Q, V←shr V)
         ELSE (DO FELÚJÍT);

Comment, a blokkállapotbit ellenőrzése
         IF (V(O,S(BC))=1) THEN (GOTO X)
                           ELSE (V(O,S(BC))←1,SZ←0);

Comment, a blokk betöltése a puffertárba
(R)      OLVAS←1, SZ←countup SZ;
         TTR←FT(FCR), PCR←Q(0),—S(BC,SZC);
         PTR←TTR, IP←1;
         PT(PCR)←PTR, IF (SZ=1)
              THEN (ADAT←PTR);
         IF (SZ≠0) THEN (PCR(8—9)←countup PCR(8—9),
              FCR(14—15)←countup FCR(14—15),
              GOTO R),
         ELSE (GOTO Z);

Comment, szókiolvasás a puffertárból
(X)      PCR←Q(0,)—S(BC,SZC);
         OP←1;
         PTR←PT(PCR);
         ADAT←PTR, GOTO Z;

Comment, mikroműveletek az olvasási művelethez
(Y)      IF (E≠0) THEN (DO FELÚJÍT);
         TTR←ADAT, IF (E≠0) THEN (PTR←ADAT);
         IR←1, IF (E≠0) THEN (IP←1);
         FT(FCR)←TTR, IF (E≠0) THEN (PT(PCR)←PTR);
(Z)      B←0, E←0, GOTO W;
         END
```

7.6.4. Teljesítőképesség értékelés

Miután megvizsgáltuk a puffertár szervezésének és működésének néhány részletét, felmerülhet az a kérdés: hogyan lehet a teljesítőképességet értékelni, és valójában mennyire is hatékony a rendszer. A teljesítőképesség értékelésének egyik útja a pufferhivatkozás kimaradásának meghatározása. A *pufferhivatkozás kimaradása* azon tár-

hivatkozások száma, amelyeket nem találtak meg a pufferben. Ez a szám egyben mutatja azt is, hogy egy blokkot hányszor vittek át a főtárból a pufferbe. Ezt a következők befolyásolják:

— a puffertár és a főtár közötti sebességarány;
— a blokkméret, a lapméret és a pufferméret;
— a tárolási algoritmus és a helyettesítési algoritmus és
— a CPU által végrehajtott programok címelrendezése.

A címelrendezés azoknak a tárcímeknek a sorrendje, amelyek a program utasításainak és adatainak végrehajtásából jön létre. Ez lehet véletlenszerű vagy sorrendi. A véletlenszerű címelrendezés olyan tárcímsorozat, amelyben bármely cím egyforma valószínűséggel fordulhat elő. A pufferhivatkozás kimaradásának valószínűsége ekkor a következőképp határozható meg: a puffer kapacitása és a főtár kapacitása közötti arány egy kis szám lesz. A blokkméretnek és a helyettesítési algoritmusoknak a variálása nincs hatással a pufferreferencia-kimaradásra. A sorrendi címelrendezés olyan tárcímsorozat, amelyben mindegyik cím egy szóval van — a sorrendben őt megelőző — cím után. A pufferhivatkozás kimaradásának valószínűsége pontosan a blokkméret inverze. A pufferméretnek és a helyettesítési algoritmusoknak nincs befolyása a pufferhivatkozás-kimaradására.

A tényleges címelrendezés sem nem véletlenszerű, sem nem sorrendi. Gibson [36] számolt be arról, hogy húsz IBM-7000 sorozatú programot alapul véve, amelyek közül mindegyik közelítőleg három millió címreferenciát futott, a pufferreferencia-kimaradás valószínűsége közelítőleg 0,0025-től 0,085-ig terjedt, és a csúcsérték 0,015 körül volt.

A puffertárszervezés hatékonyságának kiértékelésére Lipstay [37] egy módszert dolgozott ki, mely szerint először egy puffertár nélküli ideális rendszert kell definiálni, de a főtár puffertár-ciklusidőkkel működik. Az ideális rendszer azonos a puffertárból-főtárból álló rendszerrel, ha a pufferrel kiegészített rendszer CPU-ja mindig megtalálta az adatokat a puffertárban, és nincs a tárolások miatt időveszteség a tárhivatkozásokban. Tehát, az ideális rendszer a pufferrel kiegészített rendszer teljesítőképességének felső határát jelenti. Lipstay a kiértékeléshez nagymennyiségű címelrendezést alapul véve kimutatta, hogy a pufferrel kiegészített rendszer teljesítőképessége az ideális rendszerének 66...94%-a között van.

A kétszintes tárhierarchia kiértékelésének harmadik útja, Gibson [36] szerint az, hogy definiálni kell egy TR átviteli arányt:

$$TR = NT/NP, \tag{7.28}$$

ahol NT az átlagos átviteli arány a főtár és a puffertár között, NP pedig a puffertár és a CPU között. Mind az NT, mind az NP mértékegysége szó/s. A (7.28) összefüggésből következik, hogy az lenne az előnyös, ha az NP arány viszonylag nagy, az NT arány pedig viszonylag kicsi lenne. Az előbb említett IBM-7000 sorozat adatain tanulmányokat végeztek, ennek eredményeit a 7.6. táblázat foglalja össze, ahol a TR értékeket a blokkméret függvényében határozták meg. A táblázatból nyilvánvaló az, hogy a kívánt kisértékű TR átviteli arányhoz a blokkméretet kicsire és a pufferméretet nagyra kell választani. Például, 0,1-es arányt lehet kapni, ha blokkméretnek 16 szót, pufferméretnek pedig 2048 szót választunk.

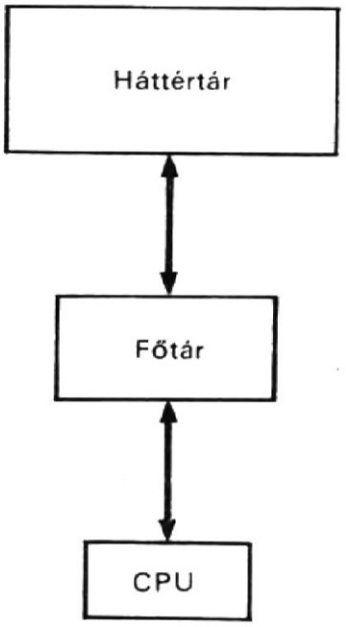
7.6. táblázat

TR átviteli sebesség arány

Blokk-méret	A puffertár kapacitása; szó								
	32	64	128	256	512	1024	2048	4096	8192
16 szó	6,5	1,9	1,4	1,0	,7	,17	,10	,045	,0033
32		12,1	3,0	2,0	1,2	,37	,16	,72	,39
64			23,2	4,5	2,5	1,2	,27	,14	,043
128				44,6	7,0	3,1	,45	,22	,073
256					84,1	11,3	2,5	,35	,09
512						157,0	6,9	,73	,20
1024							24,6	1,9	,42
2048								7,9	1,2
4096									3,7

7.7. Virtuális tár[10]

Régóta felismerték, hogy egy számítógéprendszer tárolójának sebességével és kapacitásával szemben egyidejűleg támasztott követelményeket csak hierarchikusan, egy vagy több szintű szervezéssel lehet kielégíteni. Kétszintes tár felépítése látható a 7.25. ábrán, ahol a külső tár egy vagy több tár lehet. Valamely program, adataival együtt általában a külső (háttér-) tárban van. Ha a programot vagy egy programszegmenset végre kell hajtani, beviszik a főtárba, mivel a program vagy a programszegmens utasításait csak akkor lehet végrehajtani, ha azok a főtárban vannak. Tehát, a külső tárat úgy képzelhetjük el, hogy az valamennyi számítás teljes végrehajtásához szükséges információ összességét tartalmazza. A rendszer feladata, hogy a főtár azt az információt tartalmazza, amely az adott pillanatban aktív. A számítógéprendszer feladata, nem pedig az egyes programozóké, a szabad tárterület keresése, azért, hogy az egyes programokra nézve a tárhierarchia úgy működjék, mintha csak egyetlen szintje lenne:



7.25. ábra
Kétszintes tárrendszer

[10] Denning [57] kitűnő áttekintést ad a virtuális tárról. A virtuális tárral kapcsolatos fogalmakat ebből a munkából vettük át.

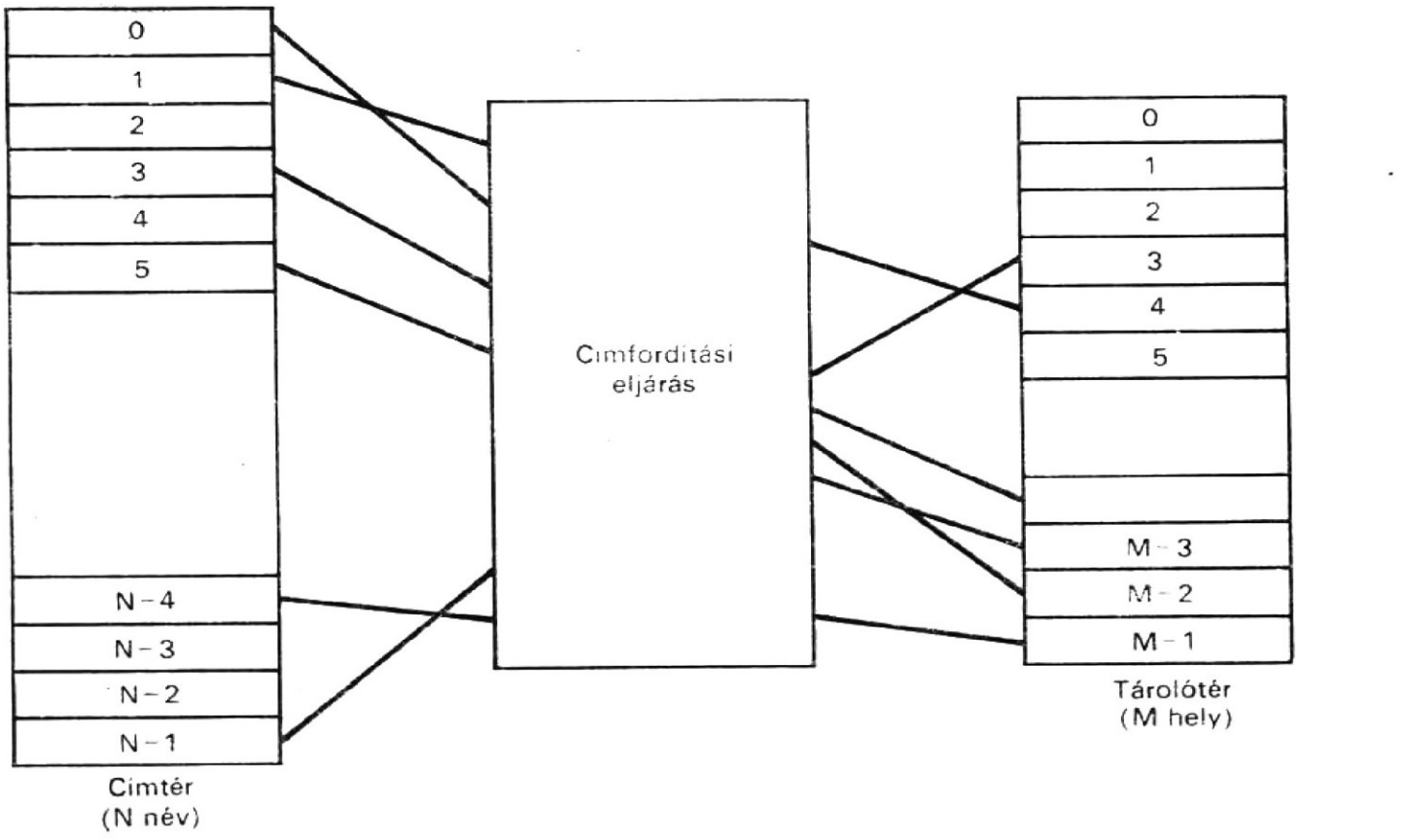
Az első időkben minden programozó maga épített be egy tárallokációs eljárást a készített programokba, ha várható volt, hogy az információ összessége meghaladja a főtár kapacitását. Az ilyen eljárás — melyet *statikus tárallokációnak* neveztek — csupán felosztotta a programot a főtár betöltéseinek (vagy szegmenseinek) olyan sorozatára, amelyek átfedték egymást. Ez azért volt lehetséges és gyakorlatilag kivitelezhető, mert a programozó jól ismerte a számítógéprendszert és a programját. A multiprogramozású és időosztásos rendszerek fejlődésével részlegesen betöltött programok futtatása vált szükségessé azért, hogy a program által felhasznált tár adatmennyiségét és a programot a tárban mozgatni lehessen a végrehajtás során. Ez azt jelenti, hogy a tárterület allokációja nem jósolható meg előre, és a szegmensátfedést előzetesen nem lehet kidolgozni. A tárallokációnak dinamikusnak kell lennie. A multiprogramozású és időosztásos rendszerekhez szükséges *dinamikus tárallokáció* vezetett el a virtuális tárak kifejlődéséhez.

Ez a fejezet bevezet a virtuális tár alapfogalmaiba, ismerteti a címfordítást, a lapfelosztást és leírja a virtuális tár ütemezését.

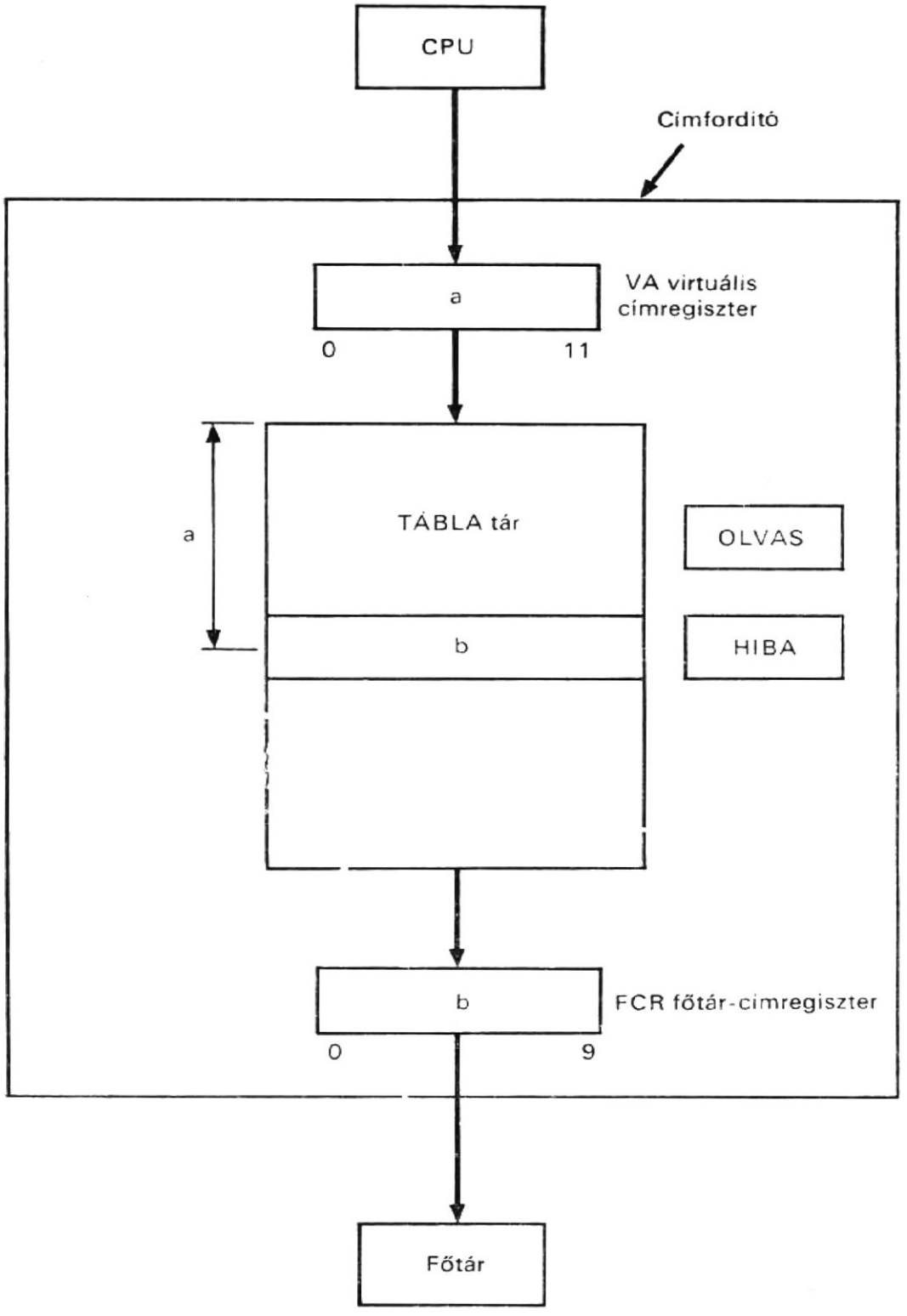
7.7.1. Alapfogalmak

Az egyszintes tárolás megvalósítását általában *virtuális tárként* ismerik. Ez a programozónak azt az illúziót nyújtja, hogy a főtár rendkívül nagy, mégha az ténylegesen viszonylag kicsi is. A számítógép hardware automatikusan viszi be az információt a főtárba, ha feldolgozás végett arra szükség van. Tehát, a virtuális tárban a tárallokálás feladata eltűnik a programból.

A virtuális tár koncepciója abból az igényből alakult ki, hogy a „cím" fogalma ne legyen azonos a „hely" fogalommal, bár a korai számítógéprendszerekben ezeket



7.26. ábra
Címfordítás a címtér és a tárolótér között



7.27. ábra.
Címfordítás realizálása

azonosaknak tekintették. A címtér nevek halmaza, amelyet egy program hivatkozási információként generálhat, míg a tártér a főtár fizikai helyeinek halmaza, ahol információt tárolnak. A Programozó által használt címet *névnek* vagy *virtuális címnek* nevezik, és ezen nevek halmaza a *címtér*. A tár által használt címet *helynek* vagy *tárcímnek* nevezik, és ezen helyek halmaza a tártér. Legyen N név a címtérben és M hely a tártérben, amint azt a 7.26. ábra illusztrálja. Több a virtuális cím, mint a tárbeli hely; tehát nem minden virtuális címnek van főtárolón belüli helye.

Ahhoz, hogy egy adott virtuális címet tárbeli címmé alakíthassunk át, szükség van egy címfordítási (transzlációs) eljárásra. A címfordítási eljárás egyik lehetséges megvalósítása látható a 7.27. ábrán. A fordítási táblázatot a TÁBLA elnevezésű tár tárolja;

a CPU virtuális címregisztere és a főtár FT tárcímregisztere (FCR) szolgál a tár TÁBLA tartalom-(puffer-) regiszteréként. A HIBA regiszter jelzi a kimaradt tételből eredő hibát. Az adott „a" virtuális címnek a főtár „b" címévé fordítását procedurális alakban a következőkben írjuk le.

```
Comment, címfordítási eljárás leírása                                    (7.29)
Register,  VA(0—11),      $a TÁBLA nevű tár címregisztere
           FCR(0—9),      $a TÁBLA nevű tár tartalomregisztere
           OLVAS,         $olvasási művelet kezdeményezése
           ÜRES,          $nincs hivatkozás a TÁBLA nevű tárra indikátor
           HIBA,          $kimaradt tétel (a szó nincs a főtárban) hibaindikátor
Memory,    TÁBLA(0—4095, 0—9), $címfordítási táblázat
           VA←„a", HIBA←0;
           OLVAS←1;
           FCR←TÁBLA(VA)
           IF (FCR=0) THEN (HIBA←0).
           END
```

A most bemutatott példában 4096 virtuális tárcím van, míg a főtárnak csak 1024 valóságos címe van. Amikor a virtuális cím nem talál tárbeli címet a TÁBLA nevű tárban, a HIBA regiszter 1 értéket kap annak jelzésére, hogy a szó nincs a főtárban. Egyébként pedig a „b" főtárbeli cím az FTR címregiszterben található.

A valóságos tárban a hely nélkül levő virtuális címek nevét a háttértárban tárolják. Ha hivatkozás történik egy ilyen virtuális címre, a HIBA regiszter 1 értéket kap. Ennek feladata a hardware vezérlése, hogy a hiányzó szót átvigye a háttértárból a főtárba. Ez szükségessé tesz:

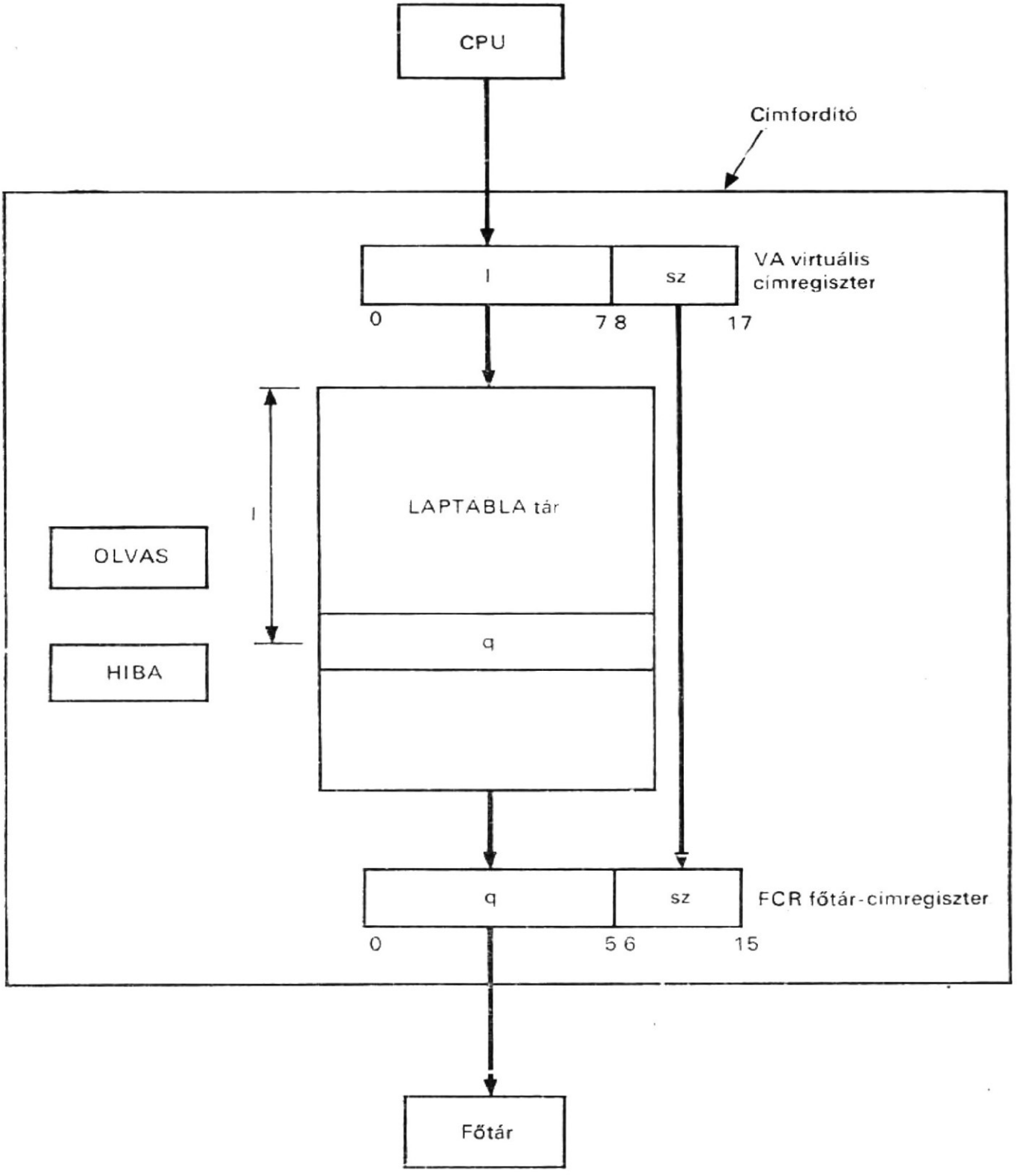
— egy *helyettesítő algoritmust*, amely eldönti, melyik szónak kell a főtárból kikerülnie;
— egy *lehívási algoritmust*, amely azt dönti el, hogy mikor kell a szót betölteni és
— egy *elhelyzési algoritmust*, amely azt dönti el, hogy hová kell a szót elhelyezni.

A címfordítási eljárás és ezek az algoritmusok a virtuális tár architektura tervezésének fontos részei.

7.7.2. Lapfelosztás

A 7.27. ábrán látható címfordítás megvalósítása nyilvánvalóan nem előnyös, mivel a TÁBLA nevű tárnak ugyanannyi szóból kellene állnia, mint a virtuális címek száma. A fordítási táblázat méretének csökkentésére a címtérben levő információt blokkokba csoportosítják. A blokk folytonos címek csoportja a címtérben. A fordítási táblázat hivatkozási helyei most blokkokra fognak hivatkozni, mivel jóval kevesebb a blokk, mint a cím.

A 7.28. ábrán a címfordítás realizálása látható, ahol a tárteret rögzített méretű, név nélküli (de címezhető) blokkokba szervezik. Ezeket a név nélküli blokkokat *lapoknak* hívják. Tételezzük föl, hogy 1024 szó van egy lapon. A fordítási táblázatot, — amelyet most *laptáblának* nevezünk — a 256 16-bites szóból álló LAPTÁBLA nevű tár tárolja. A VA(L) részregiszter tárolja a virtuális lapcímet, míg a VA(SZ) részregiszter a virtuális szócímet tartalmazza. A VA(L) részregiszter a LAPTÁBLA nevű tár címregisztere. Az FCR(Q) részregiszterben helyezkedik el a főtárbeli lapcím, míg a FCR(SZ) részregiszter a főtár szócímet tartalmazza. Az FCR(Q) részregiszter egy-



7.28. ábra
Laptechnika realizálása

úttal a TÁBLA nevű tár pufferregisztere is. Minthogy a VA(L) részregiszternek nyolc bitje van, az FCR(Q) részregiszternek viszont hat bitje, 256 virtuális lapcím van, de csak 64 memórialapcím. A fordítás egy adott „l-sz" virtuális címről egy „q-sz" tárcímre procedurális utasításokkal a következőképpen írható le:

```
Comment, a lapozási eljárás leírása                                    (7.30)
Register,   VA(0—17),        $a LAPTÁBLA nevű tár címregisztere
            FCR(0—15),       $a LAPTÁBLA nevű tár pufferregisztere
            OLVAS,           $az olvasási művelet vezérlése
            HIBA,            $kimaradtlap hibaindikátor
```

```
Subregister, VA(L,SZ)=VA(0—7, 8—17),
            FCR(Q,SZ)=TA(0—5, 6—15),
Memory,     LAPTÁBLA(VA(L))=LAPTÁBLA(0—255, 0—5),
            VA←„l-sz", HIBA←0;
            OLVAS←1;
            FCR(Q)←LAPTÁBLA(VA(L)),
            FCR(SZ)←VA(SZ);
            IF (FRC(Q)=0) THEN (HIBA←0);
            END
```

Ha a virtuális címnek nincs főtárbeli lapcíme a LAPTÁBLA nevű tárban, akkor a HIBA regiszter 1 értéket kap; egyébként a q-sz főtárcím az FCR címregiszterben van.

A laptáblázat tárolható:
— gyors, kisméretű tárolóban (7.28. ábra);
— a főtárban;
— igen gyors asszociatív tárban.

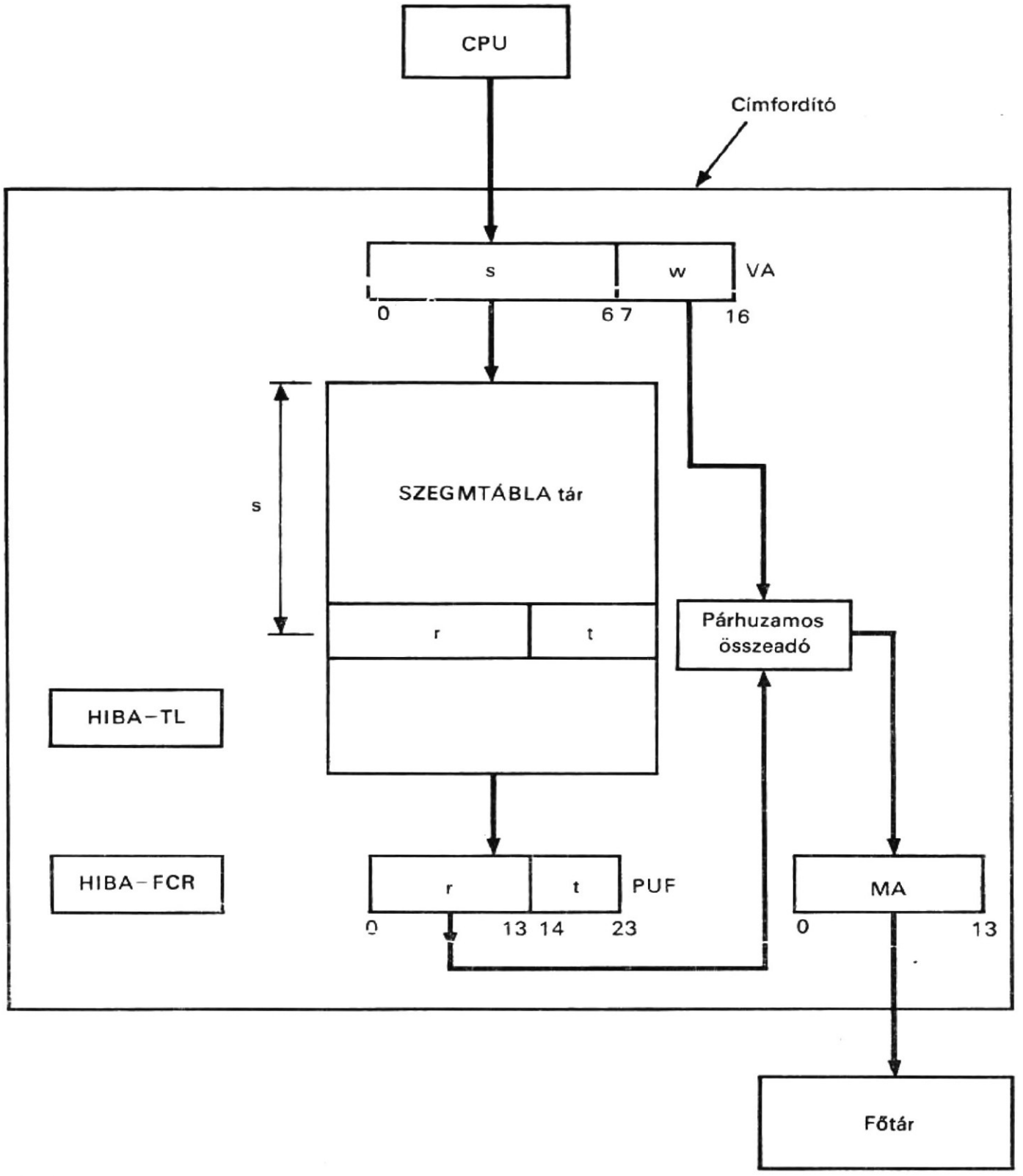
Az első esetben egy további tárra van szükség, és külön tárelérési idő tartozik egy szó eléréséhez. A második esetben két főtárciklusra van szükség, egy szó kiolvasásához tehát a program csak fél sebességgel hajtható végre (regiszter—regiszter utasítások kivételével, amelyek nem hivatkoznak a tárra). A harmadik esetben — az ár miatt — csak igen kis asszociatív tár használható gazdaságosan. Minden asszociatív regiszter egy hivatkozási helyet tárol, és csak a leggyakrabban használtakat. Ha egy adott virtuális lapcímre kell hivatkozni, csaknem azonnal előállítható a főtárbeli lapcím; egyébként pedig a főtárban levő laptáblázatra kell hivatkozást végezni. Úgy találták, hogy 8...16 asszociatív regiszter elegendő arra, hogy a programokat csaknem teljes sebességgel lehessen futtatni [49].

A laptechnikát először Angliában alkalmazták az Atlas számítógéprendszerben [41], [44], jelenleg már számos számítógéprendszerben használják, így pl. az IBM System/360 67 modellje, az RCA Spectra 70/61 és a GE 645 (és 655).

7.7.3. Szegmentálás

A címfordítás másik ismert megvalósítása látható a 7.29. ábrán, ahol a címteret tetszőleges méretű, névvel ellátott blokkokba szervezik. Ezek, a névvel ellátott blokkok a *szegmensek* A programozó számára a szegmens egy programmodul vagy egy adatstruktúra lehet. A programozó egy, a szegmentált címmezőben elhelyezkedő tételre az s szegmensnévvel és az n szónévvel hivatkozik. A szegmenst a főtár folytonos területén tárolják. Azt a tárbeli címet, ahol az s szegmens kezdődik, *báziscímnek* nevezik. A *szegmens méretét* t-vel jelölik.

Amint a 7.29. ábrán látható, a címfordítási táblázatot, amelyet most *szegmenstáblázatnak* neveznek, a SZEGMTÁBLA nevű tárban tárolják. A VA(S) részregiszter a címregiszter és a PUF regiszter a tartalom- (puffer-) regiszter. Mivel 128 szó van a SZEGMTÁBLA tárban, ezért virtuális szegmens cím is 128 van. A SZEGMTÁBLA tár mindegyik szava az r 14-bites főtárbeli címet és a t 10-bites szegmensméretet tartalmaz. Ha a w szócím kívül esik az s szegmens t méretén, mérettúllépés-hibát jeleznek ki. Minthogy előfordulhat, hogy valamely virtuális szegmens nincs a főtárban, ezeknek a szegmenseknek nem lesz hivatkozási helyük a SZEGMTÁBLA tárban; ez esetben *kimaradt szegmens hibajelzés* keletkezik. A fordítást egy adott s—w virtuális címről egy (r+w) tárcímre procedurális alakban a következőképpen írhatjuk le.



7.29. ábra
Szegmentálás realizálása

Comment, szegmentálásos elájrás leírása (7.31)

Register,	VA(0—16),	$a SZEGMTÁBLA nevű tár címregisztere
	FCR(0—13),	$a főtár címregisztere
	PUF(0—23),	$a SZEGMTÁBLA nevű tár tartalom- (puffer-) regisztere
	OLVAS,	$olvasási művelet vezérlése
	HIBA-FCR,	$kimaradt szegmens indikálás
	HIBA-TL,	$szegmens mérettúllépés indikálás

```
Subregister,   VA(S,W)=VA(0—6, 7—16),
               PUF(R,T)=PUF(0—13,14—23),
Memory,        SZEGMTÁBLA (VA(S)=SZEGMTÁBLA (0—127, 0—23),
Terminal,      B(0—12)=VA(W)*PUF(T)'+
               +PUF(T)'*B(1—3)+B(1—3)*VA(W),
               B(13)=0,
               VA←„s—w", HIBA—FCR←0, HIBA—TL←0;
               OLVAS←1;
               PUF←SZEGMTÁBLA (VA(S));
               IF (PUF(R)=0) THEN (HIBA—FCR←1, GOTO X);
               IF (B(0)=1) THEN (HIBA—TL←1, GOTO X);
               FCR←PUF(R) add VA(W);
/X/            END
```

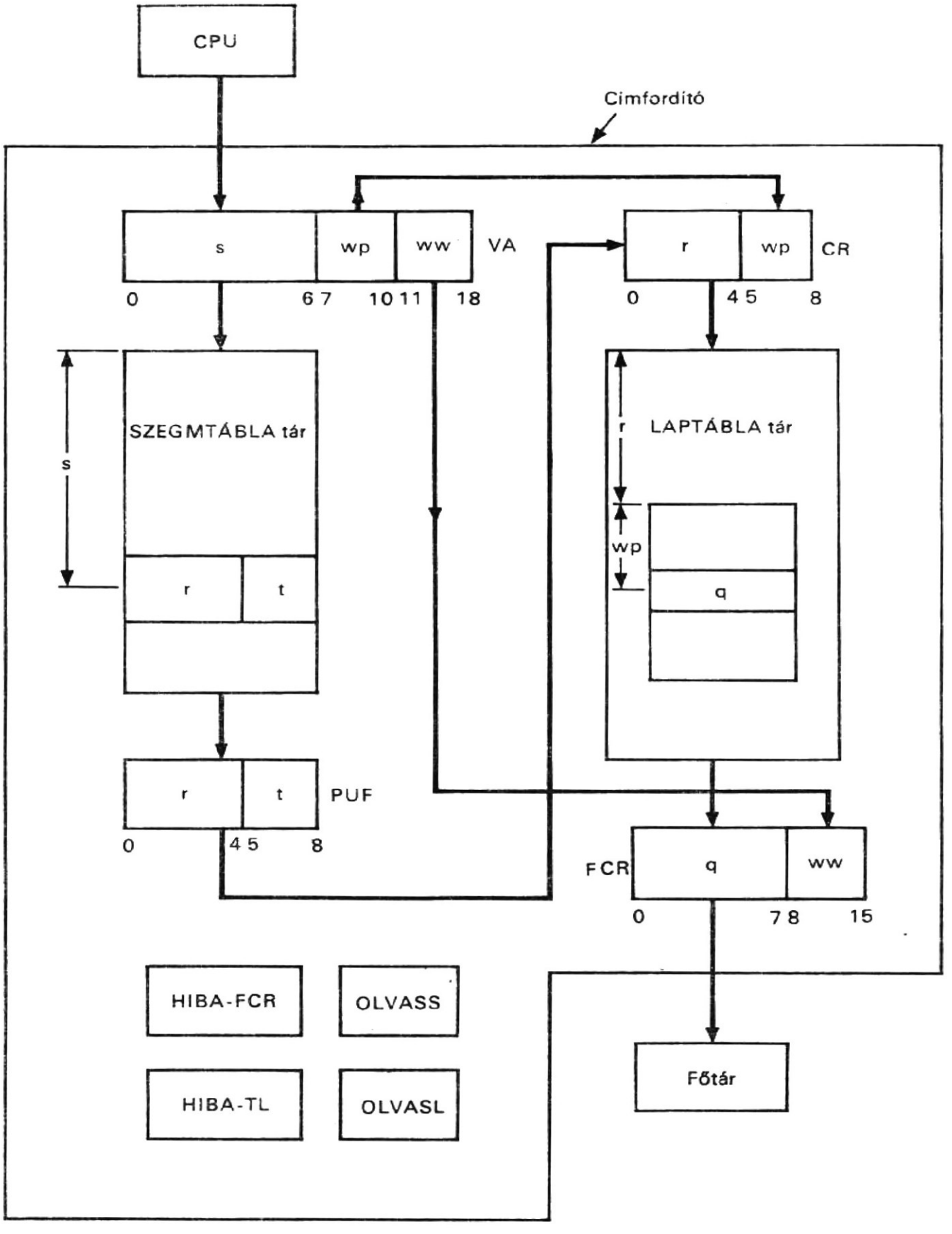
Ha a virtuális címnek nincs szegmenscíme a SZEGMTÁBLA nevű tárban, a HIBA-FCR regiszter 1 értéket kap. A fenti B terminálok annak a kivonónak a kimenetei, amely a w virtuális szócímet kivonja a t szegmensméretből. Ha a különbség negatív, a B(0) kimenet a párhuzamos kivonótól (ez nincs feltüntetve a 7.29. ábrán) 1 értéket kap; a HIBA-TL regiszter 1 értékű lesz a szegmens mérettúllépés jelzésére. Végül, a főtárbeli cím, amely az r és a w összege, az FCR főtár-címregiszterben lesz.

A lapozási megoldáshoz hasonlóan, a szegmenstáblázat tárolására még más megoldás is lehetséges. Ezt a továbbiakban nem részletezzük. Szegmentálásos megoldást használtak a Burroughs B5000 számítógép-sorozatban [42], [45] és a Rice University Computerben [43].

7.7.4. Szegmentált lapozás

Míg a szegmentálásos címfordítás a programozó igényeit elégíti ki, a lapozás a hardware-rel való kivitelezés szempontjából jelent jobb gyakorlati megoldást. A szegmentálás és a lapozás egyesíthető, amint azt a 7.30. ábrán láthatjuk. Ebben a megvalósításban minden szegmenst lapokra osztanak. Ennek megfelelően szegmenstáblákat is és laptáblákat is használunk. Minden belépési pont a szegmenstáblában egy laptáblára mutat. Egy szegmenstábla van, de annyi laptábla, ahány belépési hely van a szegmenstáblában.

Amint a 7.30. ábrán láthatjuk, a szegmenstáblázatot a SZEGMTÁBLA tár, a laptáblázatokat pedig a LAPTÁBLA tár tárolja. A VA(S) részregiszter a SZEGMTÁBLA tár címregisztere és a PUF regiszter annak tartalom regisztere. 32 szó van a SZEGMTÁBLA tárban, ami maximálisan 32 szegmens kialakítását teszi lehetővé. Egy-egy szegmensben összesen 16 lap lehet. Minden lap a főtárban egy 256 szóból álló, folytonos terület. A LAPTÁBLA tárban 512 szó van, amely összesen 32 laptáblázat létrehozását teszi lehetővé, ahol mindegyik táblázatnak összesen 16 hivatkozási helye van. Az adott virtuális cím az s 7-bites virtuális szegmenscímből, a wp 4-bites virtuális laptáblázatcímből és a ww 8-bites szócímből áll. A 7-bites szegmenscím maximálisan 128 virtuális szegmens megcímzését teszi lehetővé. Mindegyik szegmenstáblázat-hivatkozási hely az r 5-bites laptáblázatcímből és a t 4-bites szegmensméretből áll (ez utóbbi a lapok számát jelenti). A LAPTÁBLA tár szócíme az r 5-bites laptáblanév-címet és a wp 4-bites virtuális laptáblázatcímet tartalmazza. A laptáblázat mindegyik hivatkozási helyén a q 8-bites főtárbeli lapcím áll. A főtárbeli cím a q 8-bites lapcímet és a ww 8-bites szócímet tartalmazza.



7.30. ábra
Szegmentált lapozás realizálása

A címfordítás a 7.30. ábrán látható szervezés esetén a következőkben ismertetettek szerint történik. Az adott virtuális szegmenscímből megkapunk egy hivatkozási helyet a szegmenstáblában; egyébként pedig kimaradt szegmens hibajelzés történik. A szegmenstáblázat sorából megkapjuk az r laptáblázatnév-címet és a t szegmensméretet. Ha a szegmensméretet túlléptük, mérettúllépés hibakijelzés történik; egyéb-

ként pedig meghatározzuk a laptáblázatot a LAPTÁBLA-tárban az r laptáblázatnév-címmel, míg a laptáblázat bemenetét a wp laptáblázat cím határozza meg. Egy adott virtuális cím főtárbeli lapcímét a laptáblázat sorából kapjuk meg, a főtár szócímet pedig az adott ww szócímből. Ezt a fordítást egy adott s-wp-ww virtuális címből a q-ww tárcímre procedurálisan a következőképpen írhatjuk le:

```
Comment, szegmentált lapozási eljárás leírása                    (7.32)
Register,    VA(0—23),        $címregiszter
             PUF(0—8),        $a SZEGMTÁBLA nevű tár tartalom- (puffer-)
                                regisztere
             CR(0—8),         $a LAPTÁBLA nevű tár címregisztere
             FCR(0—15),       $a főtár címregisztere
             OLVASS,          $a SZEGMTÁBLA nevű tár olvasásának vezérlése
             OLVASL,          $a LAPTÁBLA nevű tár olvasásának kezdeményezése
             HIBA—FCR,        $kimaradt szegmens indikátor
             HIBA—TL,         $szegmens mérettúllépés indikátor
Subregister, VA(S,WP,WW)=VA(0—11, 12—15, 16—23),
             PUF(R,T)=PUF(0—4, 5—8),
             CR(R,WP)=CR(0—4, 5—8),
             FCR(Q,WW)=FCR(0—7, 8—15),
Memory,      SZEGMTÁBLA(VA(S))=SZEGMTÁBLA(0—127, 0—8),
             LAPTÁBLA(CR)=LAPTÁBLA(0—511,0—7),
Terminal,    B(0—3)=VA(WP)*PUF(T)'+
             +PUF(T)'*B(1—4)+B(1—4)*VA(WP),
             B(4)=0,
             VA←„s-wp-ww", HIBA—M←0, HIBA—TL←0;
             OLVASS←1;
             PUF←SZEGMTÁBLA(VA(S));
             IF (B(0)=1) THEN (HIBA—TL←1, GOTO X);
             CR←PUF(R)—VA(WP);
             OLVASL←1;
             FCR(Q)←LAPTÁBLA(CR),
             FCR(WW)←VA(WW);
             END
```

Mint a korábbiakban is, a szegmenstáblát és a laptáblát két különböző tár tárolhatja, amint az a 7.30. ábrán látható, vagy pedig mindkettő a főtárban lehet. Az utóbbi esetben három hivatkozás szükséges minden főtárbeli szó eléréséhez, ezért igen gyors asszociatív tár rendkívül fontos a szegmenstábla és laptáblák vagy legalábbis ezek egy részének tárolására.

7.7.5. Ütemezés

Mint említettük, a lapfelosztáshoz három algoritmusra van szükség:

— lehívó,
— elhelyező és
— helyettesítő algoritmusra.

Ha addig nincs betöltés, amíg hiány nem jelentkezik, akkor ezt *igényhívásnak* tekintjük. Az elhelyező algoritmus dönti el, hogy hová helyezzen egy tételt; ez gyakran fel-

használja a helyettesítő algoritmust, amely azt dönti el, hogy melyik tételt kell eltávolítani. Tehát, a legfőbb gond a helyettesítő algoritmus kiválasztása. Ez a választás főként a programok jellegétől függ.

Több általánosan ismert helyettesítési algoritmust dolgoztak ki, amelyek közül valamennyi a programok meglehetősen egyszerű viselkedését tételezi fel. Ezek az algoritmusok: véletlenszerű, az elsőnek be—elsőnek ki (**F**irst-**I**n-**F**irst-**O**ut: FIFO), a régebben felhasznált LRU (**L**east-**R**ecently-**U**sed: LRU) és az Atlas algoritmusok. A véletlenszerű algoritmus véletlen módon választja ki a helyettesítendő lapot. A FIFO algoritmus eltávolítja a legrégebben lapozott lapot. Az LRU algoritmus a legrégebben felhasznált lapot helyettesíti. A Ferranti Atlas számítógépben megvalósított Atlas algoritmus azokat a lapokat távolítja el, amelyekre feltehetőleg a leghosszabb ideig nem lesz szükség; ez hurkokból álló programoknál volt sikeres.

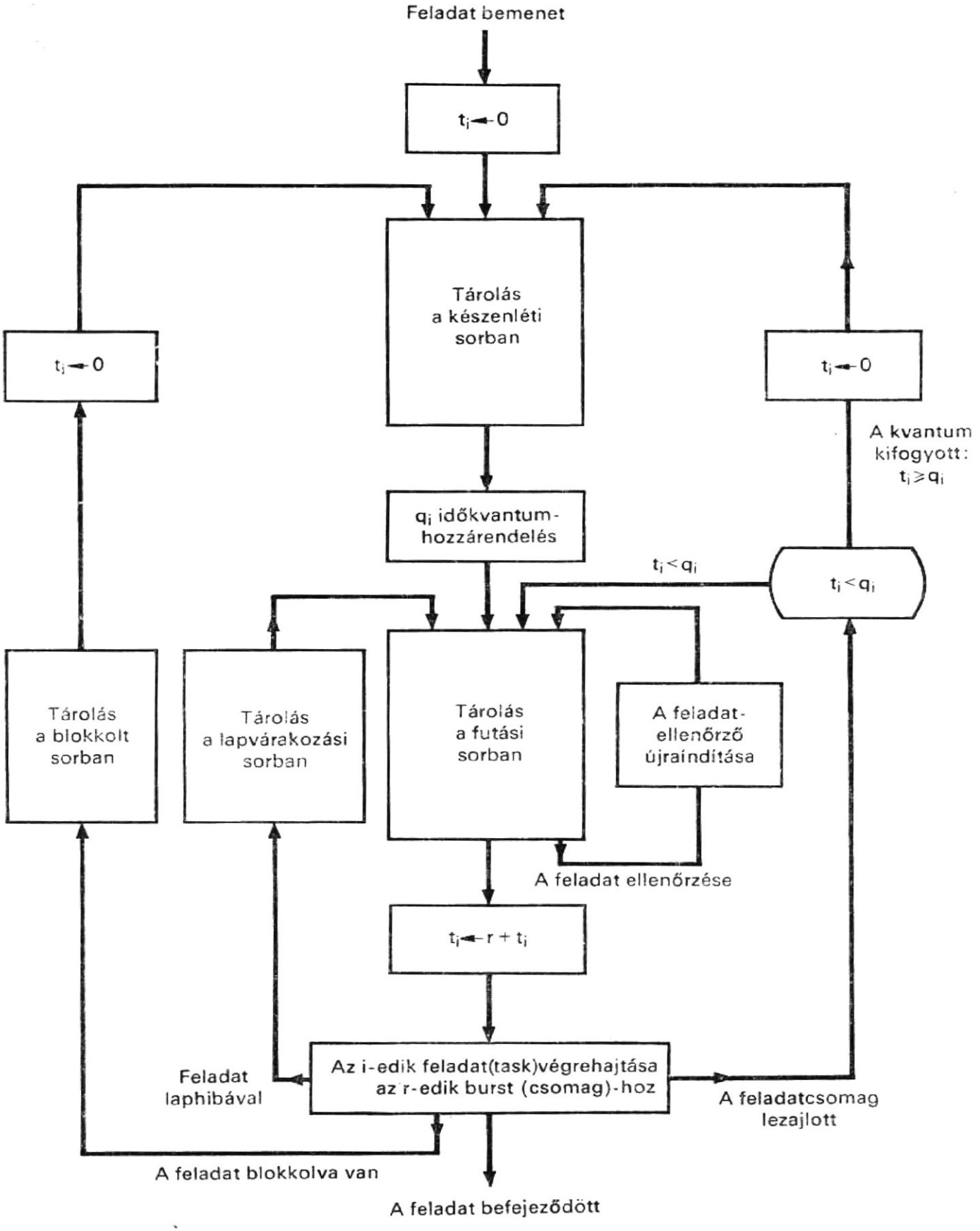
Denning [53] javasolta a működési halmaz gondolatát mint a programok viselkedését leíró modellt. Röviden, a működési halmaz azoknak a lapoknak a minimális halmaza, amelyeket be kell tölteni a főtárba ahhoz, hogy egy feladat hatékonyan működjék, felesleges laphiányok nélkül; tehát a működési halmaz szoros kapcsolatban van a feladattal (task). A helyesen megválasztott működési halmaz feltétele a lapforgalom és a felhasználatlan memóriatér közötti egyensúly. Egy feladat működési halmaza nyilvánvalóan változik az időben, és minden időpillanatban azon lapokkal definiálható, amelyekre egy kiválasztott, éppen túlhaladott időintervallumban hivatkoztak. Azok a lapok, amelyekre ezen időintervallumban nem hivatkoztak, lesznek a helyettesítés jelöltjei. Ez hasonlít az LRU módszerhez. A működési halmaz elv azonban azt állítja, hogy egy feladatot csak akkor lehet a főtárba betölteni (vagyis aktivizálni), amikor a főtár ennek működési halmaza számára rendelkezésre áll, és ha a főtár rendelkezésre állását bármely időpillanatban túllépték, a feladatot kiválasztják és eltávolítják a főtárból (azaz dezaktivizálják). A működési halmazt Denning javaslata szerint azon lapok laptáblasorainak mintavételezésével határozzák meg, amely lapok a főtárban az aktuális működési halmazok, a megválasztott időintervallumokban.

A működési halmaz modell felhasználásával javasolt Denning egy ütemezési eljárást, amely a 7.31. ábrán látható. Ebben négy sor van:

— a készenléti sor;
— a futási sor;
— a blokkolt sor és
— a lapvárakozás sor.

Akkor mondjuk, hogy egy feladat készenlétben van, ha akkor fut éppen, amikor egy processzor rendelkezésre áll. A készenléti sor tárolja az összes, készenléti állapotban levő feladatot. Egy feladatot akkor mondunk futási állapotban levőnek, ha egy processzor van hozzárendelve a feladathoz. A futási sor tárolja a feldolgozandó feladatokat. Egy feladatot akkor hívunk blokkolt állapotban levőnek, ha egy másik feladat befejezésére vár (pl. egy lapon nem levő be-, ill. kimeneti igény). A blokkolt sor tárolja ezeket a várakozó feladatokat, amíg azok nemblokkolt állapotba kerülnek, ekkor újból belépnek a készenléti sorba. A lapvárakozás sor tárolja azokat a feladatokat, amelyek arra várnak, hogy a lap bekerüljön a főtárba. Legyen i az i-edik feladat; t_i az a processzoridő, amelyet az i feladat felhasznált, amióta utoljára volt blokkolva (nem laphiány miatt); q_i legyen az i feladathoz rendelt időmennyiség; és r az a processzor-időcsomag (burst), ami alatt az i feladat egyszerre kerül végrehajtásra.

Kövessük nyomon az i feladatot a diagramon! Amikor az i feladatot létrehozzák, egy azonosítót rendelnek hozzá és elhelyezik a készenléti sorban. Feladatokat választanak ki a készenléti sorból, a domináns prioritási szabálynak megfelelően. Amint kiválasztottak egy feladatot a készenléti sorból, egy q_i időmennyiséget rendelnek hozzá, amely korlátozza a futási sorban levő idejét. A futási sor ciklikus sor; az i feladat ismételten körbekering a készenléti sorban, miközben r processzor-időcsomagokat kap



7.31. ábra
Ütemezés virtuális tárhoz

mindaddig, amíg blokkolódik vagy túllépi a q_i időmennyiséget. Ha az i feladat blokkolódik, azonosítóját elhelyezik a blokkolt sorban, amely addig ott marad, amíg meg nem szűnik a feladat blokkolása, és azután újból belép a készenléti sorba. Van egy speciális feladat, amelyet *ellenőrnek* (checker) neveznek. Ez mindig benn van a futási sorban. Az ellenőr végzi a főtár irányításának feladatát, mintákat vesz minden olyan

feladat laptábláiból, amely feladatok szolgáltatásokat vettek igénybe a legutolsó futási idejük óta, eltávolít lapokat az adott helyettesítési algoritmus szerint, és újra feltölti az üres helyeket a futási sorban oly módon, hogy feladatokat választ ki a készenléti sorból a domináns prioritási szabálynak megfelelően. Figyeljük meg, hogy az i feladat t_i idejét 0-ra állítják vissza, amikor az i feladat belép a készenléti sorba és az r csomagidővel megnövelik, amikor az i feladat továbbhalad a processzorba, végrehajtás végett.

A működési halmaz most ismertetett elvét alkalmazzák — némi módosításokkal — az RCA Spectra 70/46 időosztásos rendszerével virtuális tár tervezéséhez [55], [56].

Feladatok

7.1. Egy 4×4 elektronikus crossbarkapcsoló egyik oldalán négy tármodulhoz, másik oldalán pedig négy processzorhoz és csatornához kapcsolódik. A kapcsolóban levő adatátviteli út szélessége 36 bit és kétirányú átvitelre alkalmas. Válasszuk meg a szükséges számítógépelemeket és írjuk le CDL utasításokkal a crossbarkapcsoló struktúráját és működését!

7.2. Ahhoz, hogy a (7.4) utasításokkal leírt többszintes indirekt címzést helyesen használjuk, milyen típusú operandust lehet indirekt módon címezni?

7.3. Tervezzünk meg egy társzervezési struktúrát és azután írjuk le a többszörös indexelést, a CDL procedurális változatának felhasználásával!

7.4. Írjuk le részletesen az ebben a fejezetben a (7.8) utasításokkal megadott báziscímzés és a 9. fejezetben leírt báziscímzés közötti különbséget!

7.5. A (7.15) utasítások a push-down stack (verem) tár mélyítési műveletét írják le. Írjuk le CDL utasításokkal a pop-up stack tár műveleteit!

7.6. A (7.16) és (7.17) leírások segítségével dolgozzunk ki numerikus példát a következő feladatokra:
a) egyeztetés nagyobb, mint feltételre;
b) egyeztetés kisebb, mint feltételre!

7.7. A (7.16) és (7.18) leírások segítségével dolgozzunk ki numerikus példát a következő feladatokra:
a) egyeztetés maximumra,
b) egyeztetés minimumra!

7.8. A 7.21. ábrán bemutatott puffertár szervezésének leírásában feltételezzük, hogy nem használható a főtár első lapja.
a) Mi a jelentősége ennek a feltételezésnek?
b) Milyen változások szükségesek a pufferrel kiegészített struktúrában és a puffertár elérési sorozatában, ha megengedett az első lap használata? Készítsük el a folyamatábrát és az utasítás leírását!

7.9. Javasoljunk más helyettesítési algoritmust a 7.21. ábrán látható pufferrel kiegészített tár szervezéshez! Végezzük el a szükséges változtatásokat! Valósítsuk meg az új helyettesítési algoritmust a pufferelérési sorozatban! Készítsük el a folyamatábrát és az utasítás leírását!

7.10. Milyen fontos különbségek vannak a 7.16. és a 7.25. ábrán bemutatott kétszintes tároló hierarchiáiban?

7.11. Ha a (7.30) utasításokban leírt virtuális tár szervezéshez a laptáblát egy, nyolc regiszterből álló asszociatív tárban és a főtárban tárolják ahelyett, hogy a LAPTÁBLA tárban helyeznék el, írjuk le a lapozás megvalósítását CDL utasításokkal!

7.12. Ha a (7.31) utasításokkal leírt virtuális tár szegmentálás megvalósításában a szegmenstáblát a SZEGMTÁBLA nevű tár helyett egy nyolc regiszterből álló

asszociatív tárban és a főtárban tároljuk, írjuk le a szegmentálás kivitelezését CDL utasításokkal!

7.13. Tervezzünk mikroprogrammal vezérelt struktúrát, és mikroprogramozzuk a (7.23) és (7.24) utasításokkal leírt dinamikus betöltőt!

7.14. Tervezzünk mikroprogrammal vezérelt struktúrát, és mikroprogramozzuk a (7.25) és (7.27) utasításokkal leírt puffertárszervezést!

7.15. Mikroprogramozzuk a 7.8. feladatban leírt puffertár szervezésének vezérlését!

Irodalom

[1] CARLSON, C. B.: *The Mechanization of a Push-down Stack.* Proc. of the FJCC, Spartan Books, Inc., 1963, pp. 243...250.

[2] HELLERMAN, H.: *Digital Computer System Principles.* New York, McGraw-Hill Book Company, 1967.

[3] GSCHWIND, H.W.: *Design of Digital Computers,* Springer-Verlag New York Inc., 1967.

[4] FLORES, I.: *Computer Organization.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1969.

[5] BURNETT, G. J.—COFFMAN, F. G., JR.: *A Study of Interleaved Memory System.* Proc. of SJCC, 1970, pp. 467...474.

Betöltő

[6] CHU, Y.: *Application of Content-Addressed Memory for Dynamic Storage Allocation.* RCA Review, March, 1965, pp. 140...152.

[7] CHU, Y.: *Direct Execution of Programs in Floating Code by Address Interpretation.* IEEE Trans. on E.C., June, 1965, pp. 417...422.

[8] FLORES, I.: *Computer Software.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1965.

[9] BARRON, D. W.: *Assemblers and Loaders.* MacDonald and American Elsevier, 1969.

[10] LANZANO, B. C.: *Loader Standardization for Overlay Program.* Com. of the ACM, October, 1969, pp. 541...550.

[11] PARDO, O. R.: *A Loader Algorithm for Microprogramming.* Technical Report 70...113, Computer Science Center, University of Maryland, 1970.

[12] SLADE, A. E.—McMAHON, H. O.: *A Cryotron Catalog Memory System.* Proc. of the EJCC, December, 1956, pp. 115...120.

[13] McDERMID, W. L.—PETTERSEN, H. E.: *A Magnetic Associative Memory System.* IBM Journal of Research and Development 5, No. 1, January, 1961, pp. 59...62.

[14] FALKOFF, A. D.: *Algorithmus for Parallel-Search Memories.* J. of the ACM, October, 1962, pp. 488... ...511.

[15] ROSIN, R. E.: *An Organization of an Associative Cryogenic Computer.* Proc. of the SJCC, May, 1962, pp. 203.

Asszociatív tár

[16] DAVIES, P. M.: *A Superconductive Associative Memory.* Proc. of the SJCC, May, 1962, pp. 79.

[17] CHU, Y.: *A Destructive Readout Associative Memory.* IEEE Trans. on E. C., August, 1965. pp. 600... ...605.

[18] McKEEVER, B. T.: *The Associative Memory Structure.* Proc. of the FJCC 27, part 1, 1965. pp. 371... ...388.

[19] CAMPI, A. V.—RAY, B. H.—BUNN, R. M.: *Content Addressable Memory System Concepts.* IEEE Trans. on E. C., October, 1965, pp. 168...172.

[20] SEEBER, R. R.—LINDQUIST, A. B.: *Content Addressable Memories.* Proc. IFIP Congress, 1965, pp. 479...482.

[21] DUGAN, J. A.—GREEN, R. S.—MINKER, J.—SHINDLE, W. E.: *A Study of the Utility of Associative Memory Processors.* Proc. of the ACM, National Meeting, 1966, pp. 347...360.

[22] CHU, Y.: *A Programming Study of a Non-numerical Processor.* Technical report 56...67, Computer Science Center, University of Maryland, November, 1967.

[23] CRANE, B. A.: *Path Finding with an Associative Memory.* IEEE Trans. on E. C., July, 1968, pp. 691... 693.

[24] FELDMAN, J. A.: *An Algol-Based Associcative Language.* Comm. of the ACM, August, 1969, pp. 439...449.

[25] MINKER, J.: *An Overview and Bibliography of Associative or Content-Addressable Memory Systems.* 1956—1968, Auerbach Corporation, 1969.

[26] TAYLOR, B. T.: *Associative Memory Concepts.* unpublished notes, Computer Science Center, University of Maryland, May, 1970.

Puffer

[27] ECKERT, J. P.—CHU, J. C.—TONIK, A. B.—SCHMIT, W. F.: *Desin of UNIVAC-LARC System: I.* Proc. of the EJCC, 1959, pp. 59...65.

[28] BLOOM, L.—COHEM, M.—PORTER, S.: *Considerations in the design of a computer with high logic-to-memory speed ratio.* Proceedings of Gigacycle Computing Systems, AIEE Special Publication S-136, 1962, pp. 53...63.

[29] TAKAHASHI, S.—NISHINO, H.—YOSHIHIRO, K.—FUCHI, K.: System Design of the ETL Mk-6 Computers, Information Processing 1962, Amsterdam, The Netherlands: North Holland Publishing Co, 1963, p. 690.

[30] — —, Ferranti Computing Systems. Atlas 2, London: Ferranti Ltd., 1963.

[31] LEE F. F.: *Look-Aside Memory Implementation.* Memorandum MAC-M-99, M.I.T., Cambridge, Mass., August 19, 1963.

[32] — —: „Look-Aside Memory Simulation", Memorandum MAC-M-131, M.I.T., Cambridge, Mass., January 2, 1964.

[33] WILKES, M. V.: *Slave Memories and Dynamic Storage Allocation.* IEEE Trans. on E. C., April, 1965, pp. 270...271.

[34] SCARROTT, G. G.: *The Efficient Use of Multilevel Storage.* Proc. of the IFIPS Congress 1965, Spartan Books, 1965.

[35] BELADY, L. A.: *A Study of Replacement Algorithms for a Virtual Storage Computer.* IBM Systems Journal 5, No. 2, 1966, pp. 78...101.

[36] GIBSON, D. H.: *Considerations in Block-Oriented Systems Design.* Proc. of the SJCC, 1967, pp. 75... ...80.

[37] LIPSTAY, J. S.: *Structural Aspects of the System/360 Model 85, the Cache.* IBM Systems Journal 7, No. 1, 1968, pp. 15...21.

[38] SISSON, S. S.—FLYNN, J. J.: *Addressing patterns and memory handling algorithms.* Proc. of the FJCC, 1968, Thompson Book Company, pp. 957...967.

[39] LEE, F. F.: *Study of 'Look-Aside' Memory.* IEEE Trans. on E. C., November, 1969. 1062...1064.

[40] MATTSON, R. L.—GECESEI, J.—SLUTZ, D. R. and TRAIGER, I. L.: *Evaluation Techniques for Storage Hierarchies.* IBM Systems Journal 9, No. 2, 1970.

Virtuális tár

[41] FOTHERINGHAM, J.: *Dynamic Storage Allocation in the Atlas Computer, Including an Automatic Use of a Backing Store.* Com. of the ACM, October 1961, p. 435.

[42] FOTHERINGHAM: *The Descriptor-A Definition of the B5000 Information Processing System.* Burroughs Corporation, 1961.

[43] ILIFFE, J. K.—JODEIT, J. G.: *A Dynamic Storage Allocation Scheme.* British Computer Journal, October, 1962, pp. 200...209.

[44] KILBURN, T.—DEWARDS, D. B. G.—LANIGAN, M. J.—SUMNER, F. H.: *One-Level Storage System.* IRE Trans. on E. C., April, 1962, pp. 223...235.

[45] MACKENZIE, F. B.: *Automated Secondary Storage Management*, Datamation 11, 1965, pp. 24...28.

[46] DENNIS, J. B.: *Segmentation and the Design of Multiprogrammed Computer Systems*. J. of the ACM, October, 1965, pp. 589...602.

[47] ARDEN, B.W.—GALLER, B. A.—O'BRIEN, T. C.—WESTERVELT, F. H., *Program and Address Structure in a Time Sharing Environment*. J. of the ACM, January, 1966, pp. 1...16.

[48] BELADY, L., A.: *A Study of Replacement Algorithms for Virtual Storage Computers*. IBM Systems Journal 5, No. 2, 1966, pp. 78...101.

[49] SHEMER, J.—SHIPPEY, B.: *Statistical Analysis of Paged and Segmented Computer Systems*. IEEE Trans. on E. C., December, 1966, pp. 855...863.

[50] WILKES, M. V.: *Time Sharing Computer Systems*. New York: American Elsevier Publishing Co., 1968.

[51] RANDELL, B.—KUEHNER, C. J.: *Dynamic Storage Allocation Systems*. Com. of the ACM, May, 1968, pp. 297...305.

[52] DALEY, R.—DENNIS, J. B.: *Virtual Memory, Processes, and Sharing in Multics*. Com. of the ACM, May, 1968, pp. 306...312.

[53] DENNING, P. J.: *The Working Set Model for Program Behavior*. Com. of the ACM, May, 1968, pp. 323...333.

[54] BENOUSSAN, A.—CLONGEN, C. T.—DALEY, R. C.: *The Multics Virtual Memory*. Proc. 2nd ACM Symposium on Operating System at Princeton October, 1969, pp. 30...42.

[55] OPPENHEIMER, G.—WEIZER, N.: *Resource Management for a Medium Scale Time-Sharing Operating System*. Com. of the ACM, May, 1968, pp. 313...322.

[56] WEIZER, N.—OPPENHEIMER, G.: *Virtual Memory Management in a Paging Environment*. Proc. of the SJCC. 1969, pp. 249...256.

[57] DENNING, P. J.: *Virtual Memory*. Technical Report No. 81, Department of Electrical Engineering, Princeton University, January, 1970.

8.
Vezérlés szervezése

A vezérlőegység állítja elő a vezérlőjeleket azon egységek számára, amelyekben az előírt feladatokat el kell végezni. Egy számítógéprendszerben különböző vezérlések vannak:

— a feldolgozóegység vezérlése;
— a tárolóegység vezérlése;
— a csatorna vezérlése;
— a be-, ill. kimeneti eszközök vezérlése;
— vezérlőinformáció előállítás a gépkezelő számára.

Két módszer van, amellyel vezérlőjeleket előállíthatunk: ezeket az első két szakaszban mutatjuk be. Vezérlésre van szükség két olyan rendszeregység között is, amelyek egymással kommunikálnak; ezek rendszerint aszinkron módon működnek. Az aszinkron vezérlés elvét a 8.3. szakaszban mutatjuk be. A 8.5. szakaszban leírjuk sorrendbe állítási, címzési és prioritásos megszakítási funkciók realizálását a feldolgozóegység vezérlésében.

8.1.
Vezérlésszervezés sorrendi logikai hálózattal

A vezérlőegység jeleket állít elő egylépéses mikroműveletek, mikroművelet-sorozatok, vagy több mikroművelet-sorozat vezérlésére. A vezérlőegység által előállítandó vezérlőjelek tehát a következők:

1. egylépéses vezérlőjelek;
2. vezérlőjelek sorozata; vagy
3. többszörös vezérlőjel-sorozatok.

Ilyen vezérlőegységet sorrendi logikával vagy mikroprogrammal lehet realizálni. A huzalozott sorrendi logikai hálózattal végzett vezérlés azt jelenti, hogy logikai

áramköröket alkalmazunk a vezérlőjelek előállítására, míg a mikroprogramozott vezérlés a mikroprogramtárban tárolt programot használja fel. Ebben a szakaszban a sorrendi logikát, a következőben pedig a mikroprogramot felhasználó vezérlés szervezését mutatjuk be.

8.1.1. Egylépéses vezérlés

Egylépéses vezérlőjelek előállításához mindössze egy regiszter alkalmazása elegendő. Ilyen vezérlés szervezésére mutatunk be példát az alábbiakban.

```
Comment, egylépéses vezérlőjelek előállítása regiszterrel        (8.1)
Register,    F(0—7),
Clock,       P,
Terminal,    K(0—7)=F(0—7)*P,
```

Az F regiszterből jövő nyolc terminál (K) nyolc egylépéses vezérlőjelet ad. Az F regiszternek azon bitjei szolgáltatnak aktív kimeneteket, amelyek logikai értéke 1. Ezért maximálisan nyolc vezérlőjel jelenik meg egyidőben.

Regiszter és dekódoló kombinációját is használhatjuk hasonló célokra. Ilyen vezérlésszervezésre láthatunk példát az alábbiakban.

```
Comment, egylépéses vezérlőjelek előállítása dekódolóval          (8.2)
Register,    F(0—2),
Decoder,     K(0—7)=F,
Clock,       P,
Terminal,    M(0—7)=K(0—7)*P,
```

A dekódolóból jövő nyolc M terminál adja a nyolc egylépéses vezérlőjelet. Ebben az esetben a nyolc kimeneti kapocs (terminál) közül csak egy lehet aktív egyidőben.

8.1.2. Vezérlősorozat

Vezérlőjelek sorozatának előállításához használhatunk fázisregisztert. Az alábbiakban ilyen vezérlés realizálására mutatunk be egy példát:

```
Comment, vezérlőjelek sorozatának előállítása léptetőregiszterrel  (8.3)
Register,    F(0—7),
Clock,       P(1—2),
Terminal,    K(0—7)=F(0—7)*P(1),
/P(2)/       F←cir F,
```

Az előbbiekben feltételeztük, hogy kezdetben az F regiszternek csak egyetlen bitje tartalmaz 1-est. Amint az F regiszter tartalmát minden P(2) órajelfázisra egy bittel jobbra léptetik, K vezérlőjelek sorozata jön létre az F regiszter egymást követő bitjeiből [pl. az F(0) bitből az F(7) bit és ezután visszatérünk az F(0) bitre]. Ha elágazásra van szükség a vezérlőjelek sorozatában, a fenti ciklus léptető mikroutasítást azok a mikroutasítások kell, hogy helyettesítsék, amelyek a kívánt sorozatot és elágazást biztosítják.

A vezérlőjelek sorozatát számláló- és dekódolóáramkör kombinációval is elő lehet állítani. Ilyen vezérlési struktúrára mutatunk be példát az alábbiakban.

```
Comment, vezérlőjelek sorozatának előállítása számlálóval          (8.4)
Register,     SZ(0—2),
Clock,        P(1—2),
Decoder,      K(0—7)=SZ,
Terminal,     M(0—7)=K(0—7)*P(1),
/P(2)/        SZ←countup SZ,
```

A fenti struktúra az M(0)...M(7) terminálóktól jövő nyolc vezérlőjel sorozatát állítja elő és utána visszatér M(0)-ra, feltéve, hogy kezdetben az SZ számláló 0 volt. Ha a vezérlőjel-sorozatban elágazásra van szükség, a fenti, számlálótípusú mikroműveletet azzal kell felcserélni, amely a kívánt sorozatot és elágazást elírja; ebben az esetben a számláló speciális sorrendszámlálóvá válik, amely előírt sorrendben számlál.

8.1.3. Többszörös vezérlőjelek előállítása

Többszörös vezérlőjel-sorozat előállítására regiszter és többfázisú órajelgenerátor összekapcsolását lehet felhasználni. Az alábbiakban ilyen struktúrára mutatunk be egy példát.

```
Comment, többszörös vezérlőjelek sorozatának előállítása           (8.5)
Register,     F(1—4),
Clock,        P(1—8),
Terminal,     K1(1—8)=F(1)*P(1—8),
              K2(1—8)=F(2)*P(1—8),
              K3(1—8)=F(3)*P(1—8),
              K4(1—8)=F(4)*P(1—8),
```

A fenti struktúra négy sorozatot (K1, K2, K3 és K4) állít elő, egyenként nyolc vezérlőjellel minden sorozatban. Minthogy az órajelgenerátor fix impulzussorozatot állít elő, ezeket a vezérlőjel-sorozatokat nem lehet elágaztatni. Viszont maximálisan négyet szimultán lehet előállítani ezekből a sorozatokból.

A többszörös vezérlőjel-sorozatok előállításának másik módja a számláló és többfázisú órajelgenerátor kombinációjának alkalmazása. Ilyen struktúrára az alábbiakban mutatunk be példát.

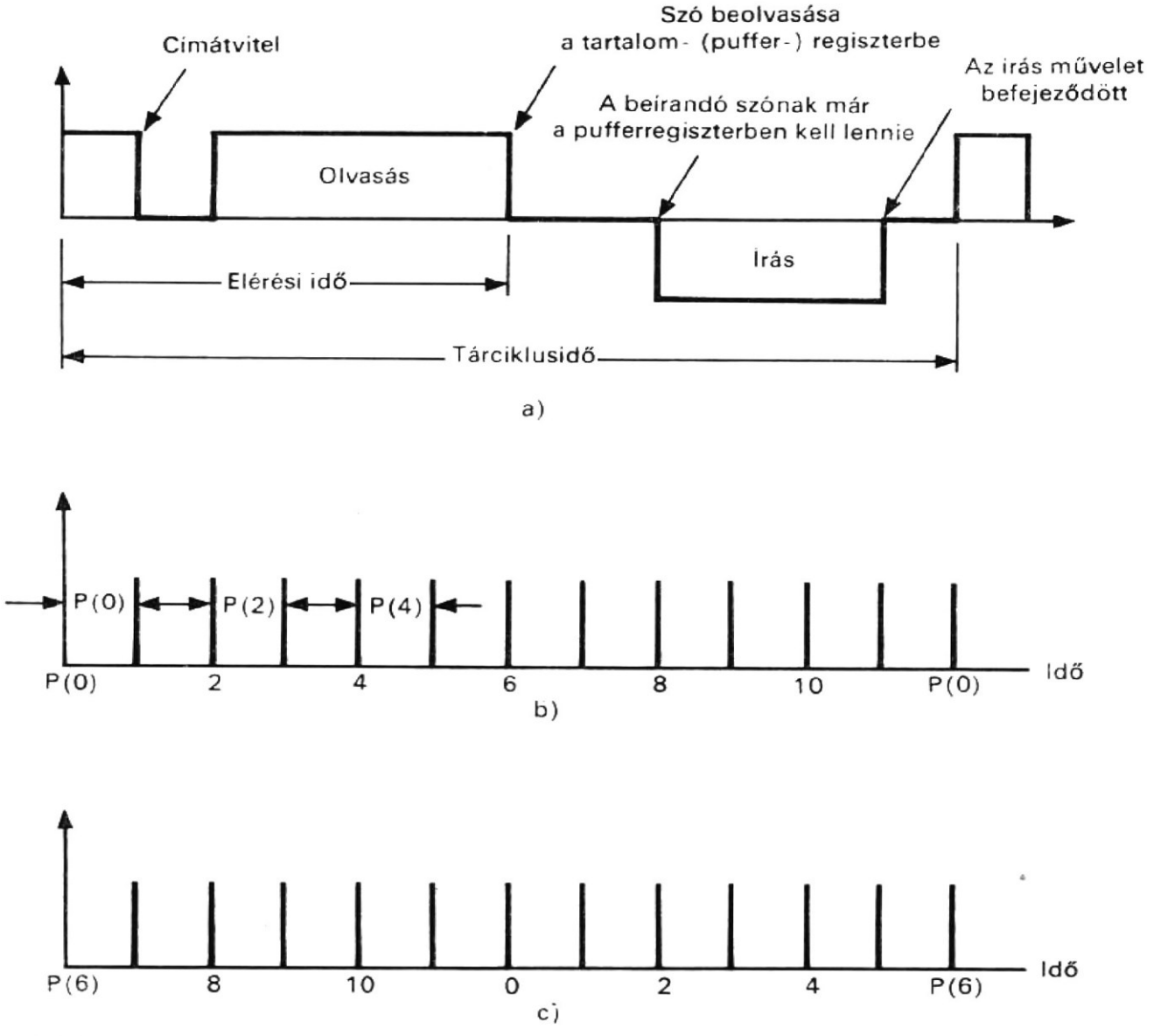
```
Comment, többszörös vezérlőjel-sorozat előállítása számlálóval     (8.6)
Register,     SZ(0—2), F(1—4),
Decoder,      K(0—7)=SZ,
Clock,        P(1—2),
Terminal,     M1(0—7)=F(1)* K(0—7) *P(1),
              M2(0—7)=F(2)* K(0—7) *P(1),
              M3(0—7)=F(3) * K(0—7) *P(1),
              M4(0—7)=F(4) * K(0—7) *P(1),
/P(2)/        SZ←countup SZ,
```

A fenti struktúra négy vezérlőjel-sorozatot (M1, M2, M3 és M4) állít elő, egyenként nyolc jellel mindegyik sorozatban. Ha elágazás szükséges ezen sorozatok valamelyikében, a fenti, számlálótípusú mikroutasítást azokkal a mikroutasításokkal kell helyettesíteni, amelyek az adott sorozat vagy sorozatok kívánt sorrendjét és elágazását adják.

8.1.4. Időzítés

A főtár ciklikusan működik. A korszerű nagyszámítógépekben a tár ciklusideje 0,5 és 2 μs közé esik. Az elérési idő az az idő, amely alatt a szó a tár tartalom- (puffer-) regiszterében (más szóval adatregiszterében) rendelkezésre áll (l. a 8.1. ábra). Ez a tárciklusidő 1/4-e és 1/2-e között van. A CPU szintén ciklikusan működik. A CPU ciklusideje — vagy órajel-periódusa — a jelenlegi számítógépekben 50 ns-től (vagy annál kisebb értéktől) 500 ns-ig terjed. Minden CPU ciklusidőben több órajelfázis van. Pl. az IBM System/360 40-es modelljének CPU ciklusideje 625 ns és egy ciklusban nyolc órajelfázis van. Az IBM System/360 85-ös modelljének CPU ciklusideje 80 ns és a ciklusban két órajelfázis van. Az RCA Spectra 70/45 központi egységének ciklusideje 480 ns és a ciklusnak négy órajelfázisa van.

Gyakorta választják meg úgy a CPU ciklusidejét, hogy az megegyezzék a főtár ciklusidejével és a CPU órajelfázisainak számát olyanra veszik, hogy illeszkedjék a logikai áramkör sebességéhez. A 8.1. ábrán példát láthatunk arra az esetre, amikor a főtár ciklusideje azonos a CPU ciklusidejével. Fontos a főtárciklusban végbemenő következő



8.1. ábra
Főtár és CPU ütemezés
a) tárciklus; *b)* CPU ciklus vagy P(0—11) órajelciklus; *c)* az órajelciklus eltolása

események ütemezése (az eseményeket azok bekövetkezési időpontjával jellemezhetjük),

a) Az az időpont, amikor a címet a tár címregiszterébe viszik át.
b) Az az időpont, amikor a tár működését kezdeményezik (ez gyakran a címátvitelt közvetlen követő idő lehet).
c) Az az időpont, amikor egy szót beolvasnak a tartalom- (puffer-) regiszterbe (vagyis az elérési idő) olvasási műveletben.
d) Az az idő, amikor a beírandó szónak már a pufferregiszterben kell lennie írási műveletben.
e) Az az idő, amikor egy szót éppen beírtak a tárba egy írási műveletben.

A fenti időzítést a 8.1*a* ábrán láthatjuk. A CPU ciklusnak 12 órajelfázisa van, amint azt a 8.1*b* ábra mutatja; tehát, a tárciklus 12 periódusra van felosztva. (A 8.1*b* és 8.1*c* ábrán a 12 órajelfázist 12 koordinátában kellene ábrázolni.) Figyeljük meg, hogy a P(i) órajel az óraperiódus kezdetén jelenik meg, de azok a mikroműveletek, amelyeket a P(i) órajel aktivizál, ezen óraperiódus vége előtt fejeződnek be. A 8.1*a*, *b* ábrán bemutatott tár és CPU időzítéséhez a tár működését a következőképpen lehet leírni.

```
Comment, a főtár leírása                                        (8.7)
Register,   C(0—14),          $címregiszter
            D(0—14),          $utasításregiszter
            R(0—23),          $tartalom- (puffer-) regiszter
Memory,     M(C)=M(0—32767, 0—23),
Clock,      P(0—11),          $CPU órajele
Comment, a CPU ciklusidő és a főtárciklusidő azonos             (8.8)
Comment, főtár olvasási művelet
/P(0)/      C←D,              $címátvitel
/P(5)/      R←M(C),           $a szó megérkezik a tartalomregiszterbe
Comment, tár olvasási utasítás
/P(0)/      C←D,              $a címátvitel
/P(10)/     M(C)←R,           $a szó tárolva van
```

Az óraciklust néha eltolják a főtárciklushoz képest. Eltolt óraciklus látható a 8.1*c* ábrán. Az alábbiakban bemutatjuk az eltolt óraciklusos tár működését.

```
Comment, tár olvasási művelet                                   (8.9)
/P(6)/      C←D,              $címátvitel
/P(11)/     R←M(C),           $a szó megérkezik a tartalom- (puffer-) re-
                              giszterhez
Comment, tár írási művelet                                      (8.10)
/P(6)/      C←D,              $címátvitel
/P(0)/
/P(4)/      M(C)←R,           $a szó tárolva van
```

A fenti eltolás készenlétbe helyezi a szót a tár tartalomregiszterében a P(0) órajel-periódus kezdetén.

8.2.
Mikroprogramozott vezérlésszervezés

A vezérlőegységet nemcsak huzalozott logika alkalmazásával, hanem a mikroprogramtárban tárolt mikroprogram segítségével is meg lehet valósítani. A mikroprogramozott vezérlés fogalmát és a mikroprogramozott számítógépek szervezését a 3. fejezetben mutattuk be. Ez a szakasz először áttekinti a mikroprogramozott vezérlésszervezést, majd bemutatja a mikroprogramozott vezérlés néhány további jellemzőjét.

8.2.1.
Mikroprogramozott vezérlőegység

A mikroprogramozott vezérlőegység, az MVE (**M**icroprogram **C**ontrol **U**nit: MCU) struktúrája a 8.2. ábrán látható. Ez az MT mikroprogramtárból, a H címregiszterből, az F tartalomregiszterből, dekódolókból, számlálóáramkörökből és elágazási logikai áramkörökből áll. A — mikroprogramtár — vezérlőtár egyes szavait *vezérlőszavaknak* vagy *mikroutasításoknak* nevezik. Ez az elrendezést részletesen leírjuk a következőkben.

Comment, mikroprogramozott vezérlőegység struktúrája (8.11)

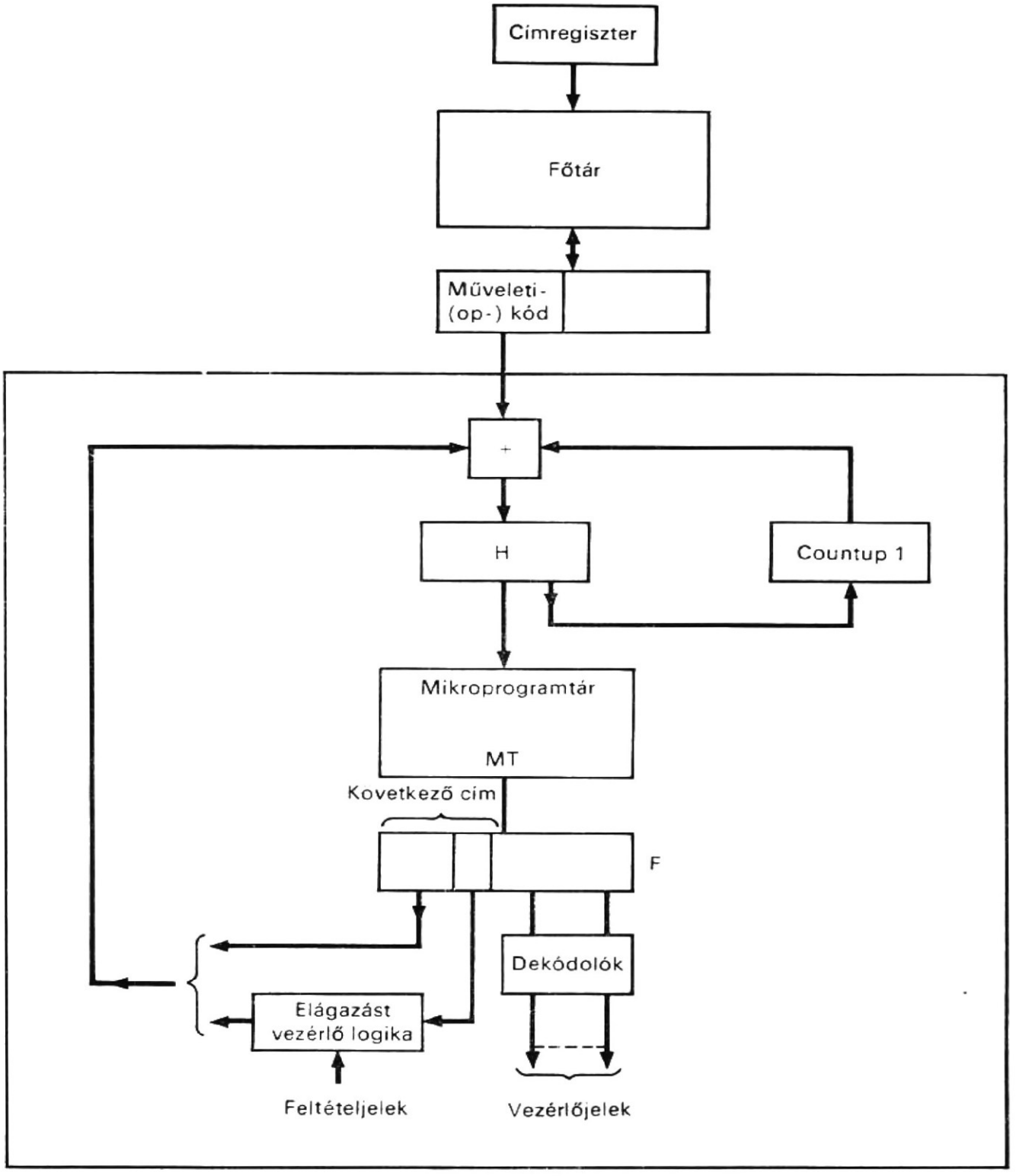
```
Register,    H(0—7),                          $címregiszter
             F(1—36),                         $mikroutasítást tartalmazó regiszter
             A(S,1—23),                       $akkumulátor
             SZ(0—3),                         $számláló
Subregister, F(ADDR)=F(0—7),                  $címmező
Memory,      MT(H)=MT(0—255,1—36),            $mikroprogramtár
Clock,       P(0—1),
/P(0)/       IF (SZ=0) THEN (H←countup H) ELSE (H←F(ADDR)),
/P(1)/       F←MT(H),
```

A mikroutasítás-regiszterben levő mikroutasítás több egylépéses vezérlőjelet tud előállítani. E vezérlőjelek kijelölése alkotja a mikroutasítás formátumát. A mikroprogramtárban levő mikroutasítások sorozata képezi a mikroprogramot; ez vezérlőjelek sorozatát állítja elő, ahol a sorozat mindegyik lépésében egylépéses vezérlőjelek jönnek létre.

A mikroprogram mikroutasításainak kívánt sorrendjét a következő mikroutasítás címének előállításával lehet biztosítani. Amint a fenti végrehajtási utasításokból látható, ezt a címet a következőképpen lehet megkapni:

a) a H címregiszter inkrementálásával;
b) a mikroutasítás címmezőjének a H címregiszterbe való átvitelével; vagy
c) a mikroutasításban levő elágazási feltételkód-mezőnek és elágazási logikai kapcsolásnak a felhasználásával, amint az a 8.2. ábrán látható.

Az elágaztatáshoz való feltételkód alkalmazása azt jelenti, hogy a következő mikroutasítás címében az alsó bit értéke 0 vagy 1, és ez a feltételkódtól függ; így ezzel kétutas elágazás hozható létre anélkül, hogy két címmezőre lenne szükség a mikroutasításban. Ha a feltételkód 3-bites és a nyolc feltételkódból hármat úgy választunk meg, hogy az alsó bitet 0 értékre állítsa, és ötöt úgy, hogy ezt a bitet 1-re állítsa, akkor ezzel



8.2. ábra
Mikroprogramozott vezérlőegység struktúrája

lehetővé tettük, hogy a címmező bármely két, szomszédos mikroprogramtárhelyen meghatározza a következő mikroutasítás címét; tehát nem lesz szükséges címinkrementálási mikroműveletre. Pontosabban a fenti kétutas elágazást a következő utasítások írják le:

Comment, példa 3-bites elágaztatási feltételkódot alkalmazó kétutas elágazásra (8.12)

Subregister, F(CC)=F(8−10) $feltételkód mező
IF ((F(CC)=0)+(F(CC)=3)+(F(CC)=5))
THEN (H(0−6)←F(0−6)H(7)←0),

IF ((F(CC)=1)+(F(CC)=4)+(F(CC)=7))
THEN (H(0—6)←F(0—6), H(7)←1),
IF ((F(CC)=2)*(A(S)=1)) THEN (H(0—6)←F(0—6), H(7)←1),

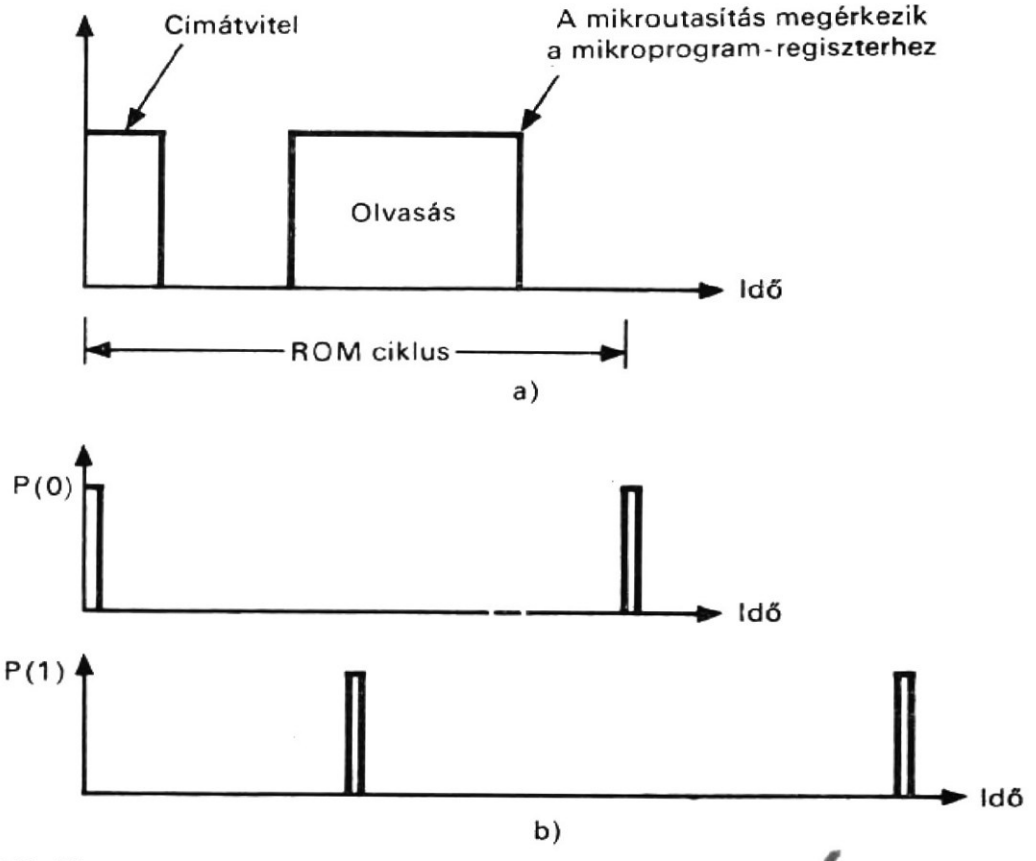
A feltételkód elágaztatásra történő felhasználása bizonyos tekintetben gátló hatású, mivel a két elágaztatott mikroutasításnak két szomszédos helyen kell lennie.

8.2.2. Időzítés

A jelenlegi mikroprogramozott számítógépek mikroprogramtárai többnyire csak kiolvasható tárat — **R**ead-**O**nly-**M**emory (ROM) — tartalmaznak. Kapacitásuk többnyire néhány ezer szó, és egy szó hossza 50 és 120 bit között lehet. Lényeges a ROM alkalmazásához az az idő, ami alatt a bekövetkező mikroutasítás címét átviszik a címregiszterbe, valamint az az idő, ami alatt a szó beolvasásra kerül a mikroutasítás-regiszterbe (azaz az elérési idő). Ezeket a 8.3*a* ábra idődiagramja illusztrálja.

Gyakran a CPU ciklusidejét úgy választják meg, hogy azonos legyen a ROM ciklusidejével. Ha a ROM ciklusidejét hosszabbra választják, a CPU a ROM sebességével fog működni. Ha pedig a ROM ciklusideje rövidebb, akkor a CPU a saját ciklusidejével működik. Mindkét esetben rosszul lesz kihasználva a hardware.

Minden ROM ciklusban egy mikroutasítást hívnak le a ROM-ból és ennek segítségével kerülnek végrehajtásra a CPU mikroműveletek. Tehát a fenti, mikroprogramozott vezérlőegységhez kétfázisú órajelgenerátort kell alkalmazni. Amint a 8.3. ábrán látható, az első fázisban a következő mikroutasítás címét írják be a H címregiszterbe és ezalatt végrehajtásra kerülnek az F mikroutasítás-regiszterben levő vezérlőbitek által aktivizált CPU mikroműveletek. A második fázisban az MT mikroprogramtárból ki-



8.3. ábra
ROM és órajel-ütemezés
a) **ROM ciklus;** *b)* **P(0—1) órajelciklus**

olvassák a következő mikroutasítást és az F mikroutasítás-regiszterbe kerül. Ez a két fázis képezi a mikroprogramozott vezérlőegység vezérlőciklusát; tehát az egyes vezérlőciklusokban a CPU mikroműveleteket egy-egy mikroutasítás vezérli.

Ha a mikroprogramtár feltételkódú elágaztatást alkalmaz, háromfázisú órajelgenerátorra van szükség. Az első fázisban átvitelre kerül a következő mikroutasítás címe. A második fázisban a mikroutasítást kiolvassák és a mikroutasítás-regiszterbe kerül. A harmadik fázisban pedig a CPU mikroműveleteket hajtják végre. Minthogy a feltételkód által specifikált feltétel mindaddig ismeretlen maradhat, amíg a CPU mikroműveleteket végre nem hajtják, a címátvitel és a CPU mikroműveletek végrehajtása nem történhet ugyanabban az órajelfázisban. Két külön órajelfázisra van szükség. Pl. az alábbiakban leírjuk egy feltételkódos elágaztatást alkalmazó mikroprogramtár műveleti sorozatát.

Comment, példa feltételkódos elágaztatású műveletsorozatra (8.13)

```
Clock,    P(0—2),
/P(0)/    IF (F(CC)=2)*(A(S)=1) THEN (H(0—6)←F(0—6), H(7)←(1),
/P(1)/    F←MT(H),
/P(2)/    CPU mikroműveletek végrehajtása
```

Ha a CPU mikroműveletekhez többlépéses vezérlőjelek szükségesek, háromnál több fázisú órajelgenerátor kell. Röviden az órajelfázisok számának megválasztása a CPU és a mikroprogramtár mikroműveleteihez szükséges vezérlőjelektől függ.

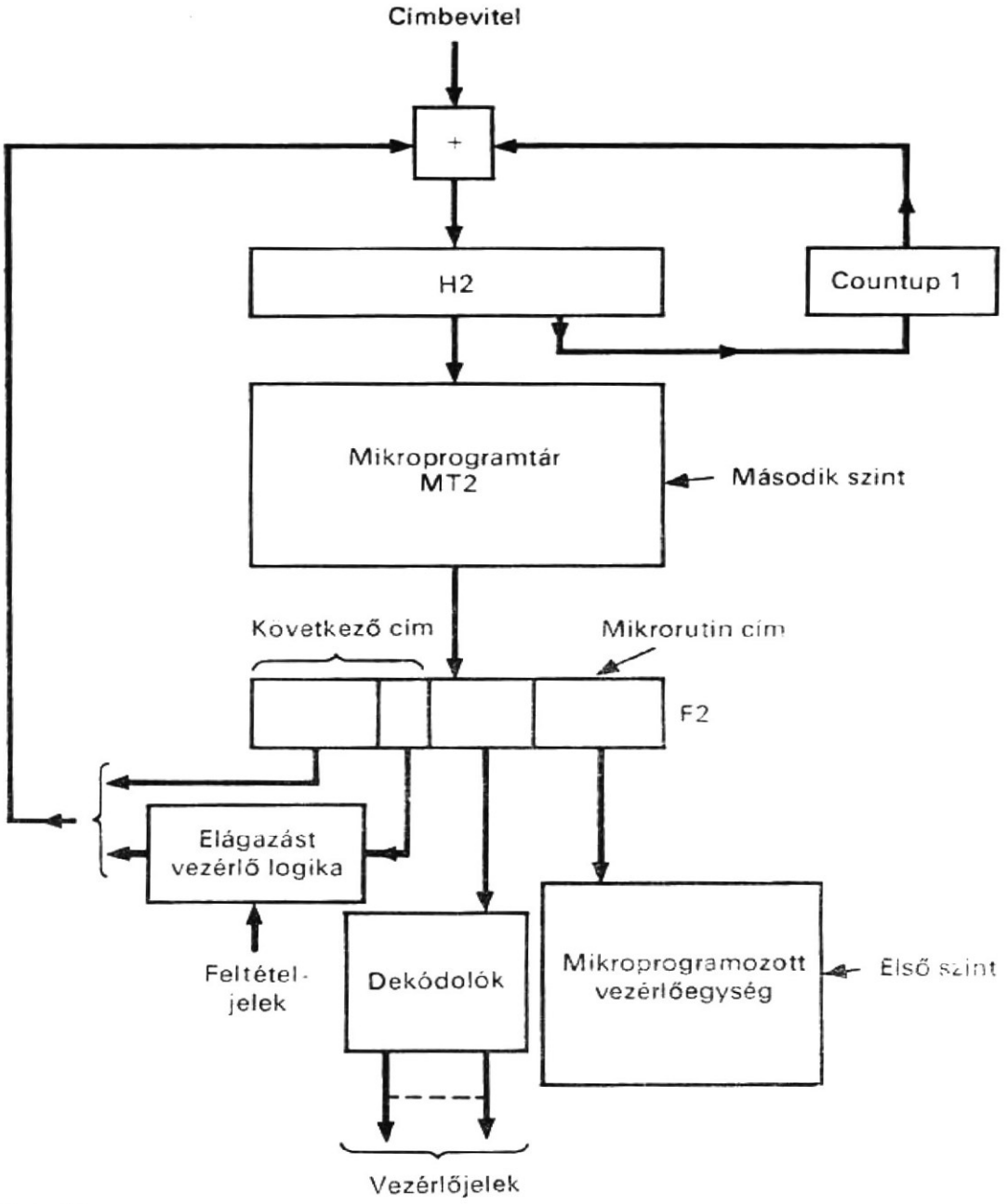
8.2.3. A vezérlés hierarchiája

A mikroprogramtárnál fontos szerepet játszanak a tár jellemzői. A technika jelenlegi fejlettségi szintjén a vezérlőtár sebessége a CPU sebességét, kapacitása pedig a mikroprogram méretét korlátozza. Valaha hasonló probléma volt a főtárral, amit a jelenleg általánosan elterjedt két- vagy többszintes tárhierarchiával hidaltak át. A tárhierarchiához hasonló vezérlési hierarchiát hozhatunk létre.

A 8.4. ábrán kétszintes vezérlési hierarchiát mutatunk be, amely két, az előbbiekben említett mikroprogramtárat használ fel. Amint a 8.4. ábrán látható, a második mikroprogramtár mikroutasításának címmezője határozza meg az elsőszintű mikroprogram címét (amelyet mikrorutinnak nevezhetünk). Ésszerű, ha feltételezzük, hogy az M1 elsőszintű mikroprogramtár gyorsabb, de kisebb kapacitású, mint az M2 másodikszintű mikroprogramtár. A gyakrabban használt mikroprogramokat a mikroutasításokban az M1 elsőszintű mikroprogramtárban tárolják, míg a „makroprogramok" a „makroutasításokban", az M2 másodikszintű mikroprogramtárban tárolják. Lehetővé kell tenni a két szint közötti kommunikációt.

A mikrorutin specifikálására egy címmező helyett két vagy több címmező szolgálhat, minthogy két vagy több mikroprogramtárat használnak. Továbbá két szint helyett lehet három vagy több szintet is alkalmazni.

A vezérlési hierarchia alkalmazásának másik módja az, hogy a számítógépet a felhasználó által mikroprogramozhatóvá teszik. Ebben az esetben az elemi mikroprogramokat az elsőszintű mikroprogramtárban tárolják. A felhasználói programokat a másodikszintű mikroprogramtárban helyezik el, amely írható-olvasható mikroprogramú tár. Ha a felhasználó rendszerprogramozó, a rendszert vezérlő mikroprogramokat a másodikszintű mikroprogramtár tárolja. Minthogy a másodikszintű mikroprogramtár gyorsabb, mint a főtár, a mikroprogramozható számítógép rugalmasabb és hatékonyabb lehet, mint a hagyományos, tárolt programú számítógép.



8.4. ábra
Kétszintes mikroprogramozott vezérlési hierarchia struktúrája

8.2.4.
A mikroutasítás formátuma

Nem egyszerű feladat a mikroutasítás formátumának kialakítása. Egyrészt a mikroutasításnak a következő funkciókat kell ellátnia:

1. a mikroutasítás vezérlőbitjeinek csoportosítása és a vezérlés kijelölése;
2. a mikroprogram mikroutasításainak sorrendbe állítása;
3. bizonyos rugalmasság elérése az újramikroprogramozáshoz.

Másrészt ezeket a funkciókat úgy kell megvalósítani, hogy minimális legyen a bitek száma a mikroutasításban, minimális legyen a szavak száma a mikroprogramtárban és minimális legyen a mikroprogram végrehajtására fordítandó idő. Ebben a fejezetben néhány olyan eljárást mutatunk be, amelyek hasznosak lehetnek a mikroutasítás kialakításához.

A vezérlőbitek mikroutasításban való kijelölésére és elrendezésére különböző eljárások léteznek:

a) a direkt vezérlés;
b) a bitcsoportosítás;
c) a többszörös formátumok;
d) a vertikális mikroprogramozás.

A direkt vezérlés azt jelenti, hogy minden vezérlőbit egy jelet generál egy mikroművelet vezérlésére. Egy nagyszámítógépben néhány száz mikroművelet lehet, amihez tehát hosszú mikroutasítás szükséges. A vezérlőbitek számát csökkenteni lehet, ha a következő három módszer közül bármelyik, vagy az összes szerint csoportosítjuk őket:

1. Azokat a mikroműveleteket vonjuk össze egy csoportba, amelyek mindig ugyanakkor fordulnak elő. Ekkor mindig csak egy vezérlőbit vezérel több mikroműveletet.
2. Azokat a mikroműveleteket tesszük egy csoportba, amelyek közül mindig csak az egyik fordul elő egyidőben. Ez a csoportosítás e mikroműveleteknek többszörös bitmezővel való kódolásával valósítható meg.
3. Több mikroműveletet egy sorozatba csoportosíthatunk és ezt a sorozatot többfázisú órajellel összekapcsolt egyetlen vezérlőbittel vezéreljük.

Ezeket a módszereket már a 3. fejezetben megtárgyaltuk. A bitek számát tovább lehet csökkenteni többszörös mikroutasítás-formátumok alkalmazásával. Ez úgy valósítható meg, hogy van egy mező, amely azt specifikálja, hogy az egyes mikroutasításokat milyen módon kell interpretálni. Minthogy a mikroutasítás rendszerint több mezőre van felosztva, a többszörös mikroutasítás-formátumot a többszörös mezőformátumokra is alkalmazni lehet. Ebben az esetben egy mezőt arra hagynak meg, hogy specifikálja, hogyan kell interpretálni a mikroutasítás egyéb mezőiben levő biteket. A többszörös formátumok alkalmazásával jelentős számú vezérlőbitet meg lehet takarítani annak árán, hogy bonyolultabb lesz a dekódolóáramkör. A vezérlőbitek számának csökkentése — bármelyiket is használjuk a fent leírt eljárásokból — azzal jár, hogy veszítünk az újramikroprogramozás rugalmasságából.

Az előbbi módszert, amikor a vezérlőbiteket használják a vezérlőjelek közvetlen generálására, *horizontális mikroprogramozásnak* nevezik. Ennek gyakori hibája a vezérlőbitek nem kellő kihasználtsága, mivel csak a vezérlőbitek kisszámú kombinációja értelmes és használható. Egy másik módszert, amely a hagyományos utasításformátum műveleti kódját és címmezőjét használja fel, *vertikális mikroprogramozásnak* nevezik. A vertikális mikroprogramozásban kívánatosabb, hogy legyen aritmetikai egység, logikai egység vagy mindkettő, amelynek műveleteit azonnal vezérelni tudják a műveletikód-mezőben levő műveleti kódok. Ehhez az egységhez az operandusokat rendszerint néhány regiszterben tárolják; tehát, három további mező áll rendelkezésre a két operandusregiszter és az eredmény tárolására szolgáló regiszter kijelölésére. Ahhoz, hogy valamelyik operandus egy adott konstans lehessen, a konstans tárolásához egy speciális mezőre, az ún. *emitmezőre* van szükség. Az emitmezőben levő konstanst a hardware viszi át a kijelölt helyére.

A mikroutasítás második fontos funkciója a mikroprogramban levő vezérlőszavak sorrendbe állítása, elágaztatása és hurok kialakítása. E funkciót többféleképpen lehet végrehajtani:

a) az aktuális cím inkrementálása;
b) a következő mikroutasítás címének felhasználása;
c) az elágaztatási feltételkód felhasználása;
d) az elágaztatási forráskód felhasználása;
e) számláló- vagy állapotregiszter alkalmazása.

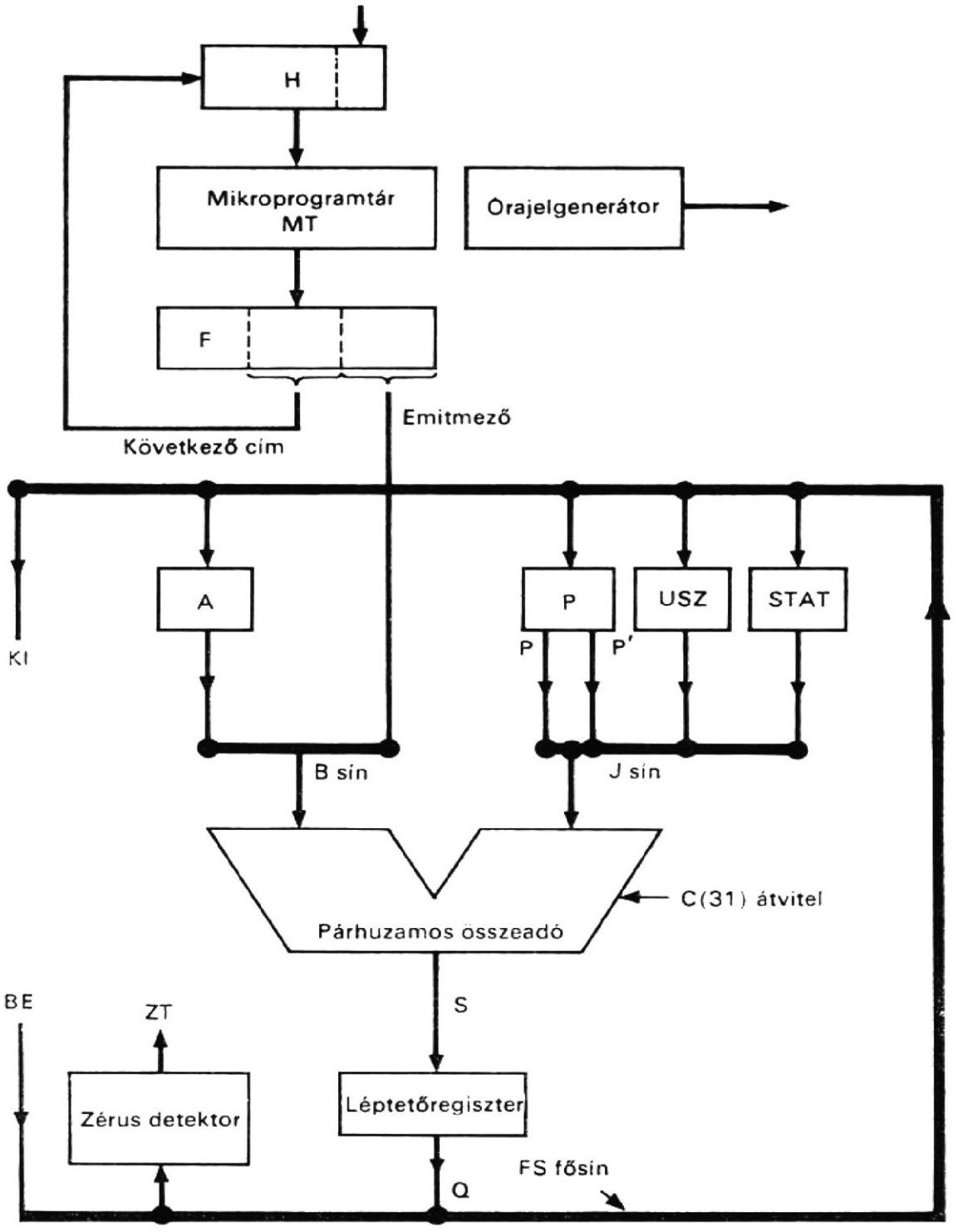
A legegyszerűbb megoldás a következő mikroutasítás címének kialakításához a címregiszterben levő aktuális cím inkrementálásával végzett sorrendbe állítás; nem igényel vezérlőbitet. Előnyösebb azonban, ha a sorrendbe állításhoz a mikroutasítás egyik címmezőjében specifikáljuk a következő mikroutasítás címét, mivel a mikroprogram hatékonyabban készíthető el, ha az egymást követő mikroutasítások a mikroprogramtár tetszőleges helyén lehetnek. Ez továbbá gazdaságos megoldást kínál az elágaztatásra, amint azt a következőkben megfigyelhetjük. Kétutas elágaztatáshoz rendszerint két következő címre van szükség. Azonban csak egyetlen következő mikroutatítás-tárcím kell, ha ennek a következő mikroutasításcímnek a kishelyiértékű bitjét valamilyen elágaztatási feltételkód szerint lehet módosítani, amint azt az előzőekben, a (8.12) utasításban már illusztráltuk.

Ahelyett, hogy egyetlen kishelyiértékű bit lenne a kétutas elágaztatáshoz, használhatunk két kishelyiértékű bitet négyutas elágaztatáshoz, három kishelyiértékű bitet nyolcutas elágaztatáshoz és így tovább. Ezen felül a kishelyiértékű bitek helyett felhasználhatók a nagyhelyiértékű bitek is többutas elágaztatáshoz. A feltételkódmező alkalmazása többutas elágaztatáshoz mindössze néhány bitet kíván, de az elágaztatás korlátozott, mivel az elágazott mikroutasítások a mikroprogramtár meghatározott helyeire vannak beszorítva. A fenti elágaztatást hatékonyabbá lehet tenni, ha pótlólag még elágaztatási forráskódmezőt is alkalmazunk. Az elágaztatási forráskód módot ad annak megválasztására, hogy a következő mikroutasítást a címmezőjében levő címről, vagy pedig valamilyen más forrásból, mint pl. egy másik regiszterben levő címről vegyünk. Tehát ezzel az eljárással többutas elágaztatást hozhatunk létre. Másfelől elágaztatást kaphatunk állapot- (fázis-) regiszter alkalmazásával, amely az adott feltételt tárolja, amikor az rendelkezésre áll és azután később elágaztatásra használják fel. Ezzel az eljárással elkerülhető a mikroutasítások megkétszerezése, mielőtt az elágaztatásra ténylegesen sor kerülne. Amellett mikroutasításokat is megtakarít, mivel a sorrendinformáció a mikroprogramtáron kívül őrződik meg. A programhurok kialakításához számlálóra van szükség. A mikroprogramozott hurokból a kilépést úgy lehet végrehajtani, hogy elágaztatást hozunk létre, amikor a számláló elér egy bizonyos értéket. A hurok — mindannyiszor, amikor felhasználásra kerül —, mikroutasításokat takarít meg. Vezérlőbitek is megtakaríthatók, mivel a sorrendinformációt kívül őrzik és nem a mikroutasításban.

Az említett eljárás felhasználását a mikroutasítás formátumának kialakításában az alábbiakban, egy mikroprogramozott CPU és egy mikroprogramozott B/K — I/O — vezérlőegység leírásával illusztráljuk.

8.2.5. Mikroprogramozott CPU

A mikroprogramozott CPU [12] struktúrája a 8.5. ábrán látható. A CPU az A akkumulátort, a B tartalomregisztert (pufferregisztert), az USZ utasításszámlálót, a STAT állapot- (státusz-) regisztert, párhuzamos összeadót, léptetőregisztert és ZT zérusdetektort tartalmaz. Továbbá szerepel az FS fősín, a B sín a párhuzamos összeadó bal oldali bemeneteihez, és a J sín a párhuzamos összeadó jobb oldali bemeneteihez. Legyen S a párhuzamos összeadó kimeneti terminálja, Q a léptetőregiszter kimeneti terminálja, BE a bemeneti terminálok, és KI a kimeneti terminálok szimbolikus jele. A párhuzamos összeadó két operandust tud összeadni a C(31) bemeneti átvitellel, amely 0 vagy 1 lehet. A struktúrát a következő utasítások írják le.



8.5. ábra
CPU struktúrája

```
Comment, mikroprogramozott CPU struktúrája
Comment, CPU struktúrája                                        (8.14)
Register,    A(0—15),          $akkumulátor
             P(0—15),          $tartalom- (puffer-) regiszter
             USZ(0—15),        $utasításszámláló
             STAT(0—15),       $vezérlőregiszter
Bus,         FS(0—15),         $fősín,
             B(0—15),          $az ALU bal oldali bemeneti sínje
             J(0—15),          $az ALU jobb oldali bemeneti sínje
Comment, bemeneti és kimeneti terminálok
Terminal,    BE(0—15),
             KI(0—15),
```

Comment, a zérusdetektor (teszter) leírása (8.15)

Terminal, $ZT = FS(0)' * FS(1)' * FS(2)' * \ldots * FS(15)'$,

Comment, a párhuzamos összeadó leírása (8.16)

$$S(0-31) = B(0-31) \oplus J(0-31) \oplus C(0-31),$$
$$C(0-30) = B(1-31) * J(1-31) + J(1-31) * C(1-31) + C(1-31) * B(1-31),$$

A 8.5. struktúrában nagyszámú lehetséges adatút van. Ezeket az utakat az *összekapcsoló (connect) mikroműveletek* írják le, ilyen pl. a Q léptetőregiszter-kimenet összekapcsolása az FS fősínnel (FS=Q), vagy az S kapocspárok (terminálok) összekapcsolása a Q-val, de ez egybites balra léptetéssel történik (Q=shl S). Ezt írja le az a mikroművelet, amelyben a jelek átvivődnek az FS fősínen keresztül az A regiszterbe (A←FS). [A kapocspárok (terminálok) vezérelt összekapcsolását hardware valósítja meg és ezt vezérelt kapuk végzik. Az eljárást *kapuzásnak* (gating) is nevezhetjük, ami azt jelenti, hogy az adatátvitelt különböző adatutakhoz lehet kapuzni]. Ezeket a mikroműveleteket a kijelölt mezőkben csoportosítva és kódolva lehet leírni, ahol a mező egy kódja rendszerint egy-egy ilyen mikroműveletet jelöl ki. A kétbites CB mező vezérli azokat az összekapcsolásokat, amelyek a B sínnel vannak kapcsolatban, ezek pontos értelmezése a 8.1. táblázatban látható. A 3-bites CJ mező jelöli ki azokat a csatlakozásokat, amelyekben a J sín vesz részt (8.2. táblázat). A 2-bites COP mezőt választottuk a párhuzamos összeadó mikroműveleteinek kijelölésére, mint ezt a 8.3. táblázat mutatja. A 2-bites CLF mező jelöli ki a léptetőregiszter lehetséges kapuzott összekötéseit (8.4. táblázat). A 3-bites CFS mező jelöli ki azokat a csatlakozásokat, amelyekben az FS fősín vesz részt (8.5. táblázat). A mikroprogramtárnak 8-bites címe

8.1. táblázat

CB mező

Kód	Kapuzott összekötés mikroművelet
00	B=0
01	B(0—7, 8—15)=0—F(EM),
10	B(0—7, 8—15)=F(EM)—0,
11	B=A

8.2. táblázat

CJ mező

Kód	Kapuzott összekötés mikroművelet
000	J=0,
001	J=P,
010	J=P′,
011	J=USZ,
100	J=STAT,
101	nem használják
110	
111	

8.3. táblázat

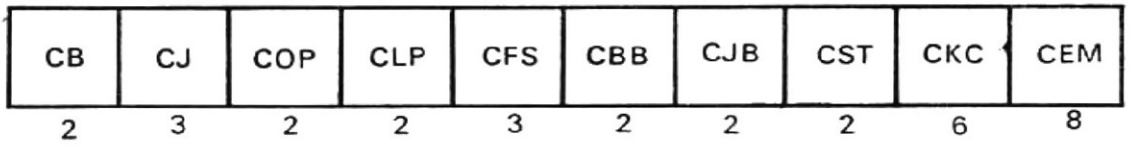
COP mező

Kód	Operátor
00	összeadás, C(31)=0
01	összeadás, C(31)=1
10	nem használják
11	

8.4. táblázat

CLF mező

Kód	Kapuzott összekötés mikroművelet
00	FS=Q, Q=S,
01	FS=Q, Q=shr S,
10	FS=Q, Q=shl S,
11	FS=BE,



8.6. ábra
A mikroprogramozott CPU mikroutasításának mezői

van. Közülük a H(0—5) nagyobb helyiértékű biteket a CKC következő címmező szolgáltatja, míg a két kisebb helyiértékű bitet H(6,7) a CBB és CJB mezők határozzák meg, amint az a 8.6. és 8.7. táblázatban látható. A zérusdetektor (teszt) eredményének és az előjelbitnek a STAT regiszterben való tárolását a CST mező jelöli ki (8.8. táblázat). A CEM emitmező egy 8-bites konstans tárolására szolgál. Ezek a mezők a 8.6. ábrán láthatók. A mikroutasítás-mezőket a 8.9. táblázat foglalja össze.

8.5. táblázat

CFS mező

Kód	Mikroművelet
000	nincs művelet
001	A←FS,
010	P←FS,
011	USZ←FS,
100	STAT←FS,
101	KI=FS,
110	nem használják
111	

8.6. táblázat

CBB mező

Kód	Mikroművelet
00	H(6)←0,
01	H(6)←1,
10	H(6)←STAT(0),
11	H(6)←STAT(2),

A mikroprogramozott vezérlés struktúrája az MT mikroprogramtárat, a H címregisztert, az F mikroutasítás-regisztert, háromfázisú órajelgenerátort és nyolc dekódolót tartalmaz. Ez utóbbi a mikroutasítás nyolc kódolt mezőjének dekódolását végzi. A struktúrát az alábbiakban írjuk le.

8.7. táblázat

CJB mező

Kód	Mikroművelet
00	H(7)←0,
01	H(7)←1,
10	H(7)←STAT(1),
11	H(7)←B(0),

8.8. táblázat

CST mező

Kód	Mikroműveletek
00	nincs művelet
01	IF (ZT=0) THEN (START(0)←0) ELSE (STAT(0)←1),
10	STAT(1)←FS(0),
11	IF (ZT=0) THEN (STAT(0)←0) ELSE (STAT(0)←1), STAT(1)←FS(0)

8.9. táblázat

A mikroprogramozott CPU mikroutasítás formátuma

Mező jele	Bitpozíciók	Jelentés
CB	0—1	A B sínhez csatlakoztatható regiszterek vezérlése
CJ	2—4	A J sínhez csatlakoztatható regiszterek vezérlése
COP	5—6	Az ALU műveleti kódja (op-kód)
CLP	7—8	A léptetőregiszter műveleti kódja
CFS	9—11	Az FS sínhez csatlakoztatható regiszterek vezérlése
CBB	12—13	Elágaztatási kód a H(6) bithez
CJB	14—15	Elágaztatási kód a H(7) bithez
CST	16—17	Kód a STAT regiszterhez
CKC	18—23	Következő mikroutasításcím
CEM	24—31	Emitmező

Comment, mikroprogramozott vezérlés struktúrája (8.17)

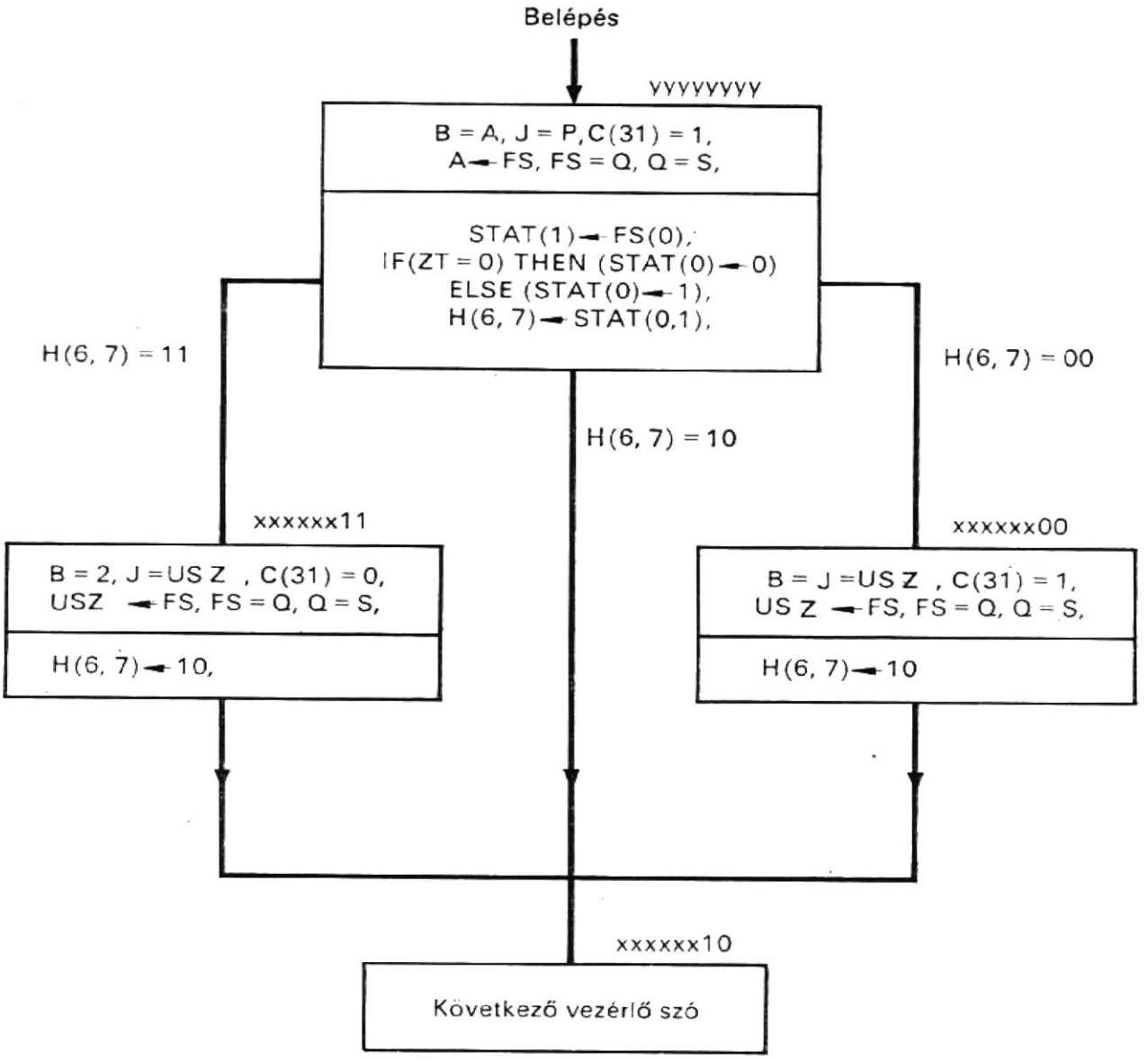
```
Memory,      MT(H)=MT(0—255, 0—31)   $mikroprogramtár
Register,    H(0—7),                 $címregiszter
             F(0—31),                $mikroutasítás-regiszter
Subregister, F(CKC)=F(18—23),
Clock,       P(0—2),
Decoder,     KB(0—3)=F(0—1),         $a CB mező dekódolója
             KJ(0—4)=F(2—4),         $a CJ mező dekódolója
             KOP(0—1)=F(5—6),        $a COP mező dekódolója
             KLP(0—3)=F(7—8),        $a CLP mező dekódolója
             KFS(0—5)=F(9—11),       $a CFS mező dekódolója
             KBB(0—3)=F(12—13),      $a CBB mező dekódolója
             KJB(0—3)=F(14—15),      $a CJB mező dekódolója
             KST(0—3)=F(16—17),      $a CST mező dekódolója
```

Egy mikroprogram leírásának és elkészítésének illusztrálása végett vizsgáljuk meg azt a műveletsorozatot, amely összehasonlítja az A és B regiszterben levő számot, feltételezve, hogy ezek a számok 2-es komplemensben vannak ábrázolva. Ha az A regiszterben levő szám nagyobb a B regiszterben levőnél, nem történik változás az USZ utasításszámlálóban. Ha a két szám egyenlő, az USZ regiszter értékét eggyel megnövelik. Ha az A regiszterben levő szám kisebb, mint a B regiszteré, az USZ regiszter tartalmát kettővel növelik meg. Ezt az összehasonlítást úgy végzik el, hogy kivonják a B regiszterben levő számot az A regiszterben levő számból, majd ezután megvizsgálják a ZT zárusdetektor (teszt) kimenetének értékét és az FS(0) fősínbitet. Ha a ZT és FS(0) értéke 10, 00 vagy 11, akkor az A regiszterben levő szám nagyobb, egyenlő, ill. kisebb, mint a B regiszterben levő szám (8.10. táblázat).

8.10. táblázat

A zérusdetektor (teszt) értelmezése

ZT	FS(0)	Értelmezés
0	0	az S összeg nulla
0	1	nem lehetséges
1	0	az összeg pozitív
1	1	az összeg negatív



8.7. ábra
Folyamatábra elágaztatási sorozathoz

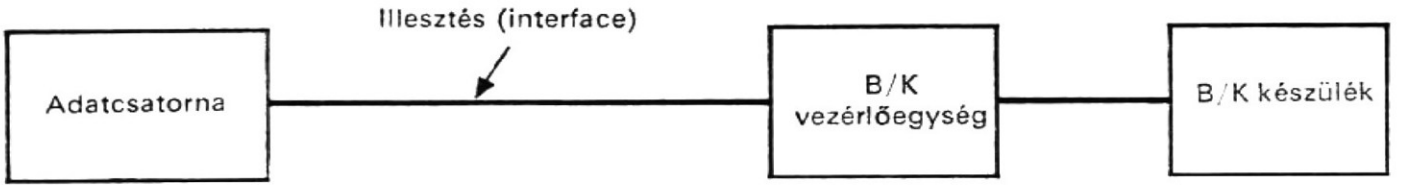
E műveletsorozat folyamatábrája a 8.7., amelyben három blokk van, egy-egy mindegyik mikroutasításhoz. Az egyes blokkokban a felső rész specifikálja a vezérelt összeköttetési és átviteli mikroműveleteket; az alsó rész határozza meg a következő mikroutasítás címét. Az egyes blokkok fölött levő szám a blokk által specifikált mikroutasítás hozzárendelt címe. Az „yyyyyyyy" címmel ellátott blokk hajtja végre a fentiekben leírt összehasonlítást; a kivonás úgy történik, hogy a B regiszterben levő szám 2-es komplemensét hozzáadják az A regiszter tartalmához. Az elágazás úgy megy végbe, hogy a következő mikroutasítás címének H(6,7) bitjeit a STAT(0,1) bitekből kapják meg, amely a ZT és FS(0) értéket tárolja. Az „xxxxxx00" címmel ellátott blokk eggyel megnöveli az USZ regiszter tartalmát, míg az „xxxxxx11" címmel rendelkező blokk kettővel növeli meg az USZ regiszter tartalmát. Tekintet nélkül arra, hogy az összehasonlítás eredménye pozitív, nulla vagy negatív, a sorozat végül is a mikroutasítást az „xxxxxx10" címen éri el.

A fenti három mikroutasítást a következő utasításokkal írjunk le. Megfigyelhető, hogy rendelkezésre áll a START kapcsoló és a G vezérlőregiszter, amelyek feladata a sorozatnak egy, a H regiszterben levő, megfelelő címről való indítása.

```
Comment, mikroprogram az elágazással rendelkező műveletsorozathoz      (8.18)
Comment, a sorozat indítása
/START(ON) * P(0)/          F←0, G←0,
```

```
/G'*P(1)/          H←„yyyyyyyy", G←1
/G*P(2)/           F←MT(H)

Comment, az yyyyyyyy címen elhelyezett mikroutasítás a B-nek A-ból való
                   kivonására

/KB(3)*P(0)/       J=A,
/KJ(1)*P(0)/       B=P,
/KOP(1)*P(0)/      C(31)=1,
/KLP(0)*P(0)/      FS=Q, Q=S,
/KFS(1)*P(0)/      A←FS,                                              (8.19)
/KST(3)*P(0)/      IF (ZT=0) THEN (START(0)←0) ELSE (STAT(0)←1),
                   STAT(1)←FS(0),
/KBB(2)*P(1)/      H(6)←STAT(0),
/KJB(2)*P(1)/      H(7)←STAT(1),
/G*P(1)/           H(0—5)←F(CKC),    $F(CKC)=„xxxxxx"
/G*P(2)/           F←MT(H),

Comment, az xxxxxx00 helyen elhelyezett mikroutasítás, az USZ-nek 1-gyel
                   történő növelésére

/KB(0)*P(0)/       B=0,
/KJ(3)*P(0)/       J=USZ,
/KOP(1)*P(0)/      C(31)=1,
/KLP(0)*P(0)/      FS=Q, Q=S,
/KFS(3)*P(0)/      USZ←FS,                                            (8.20)
/KST(0)*P(0)/
/KBB(0)*P(1)/      H(6)←1,
/KJB(1)*P(1)/      H(7)←0,
/G*P(1)/           H(0—5)←F(CKC),    $F(CKC)=„xxxxxx"
/G*P(2)/           F←MT(H)

Comment, az xxxxxx11 helyen elhelyezett mikroutasítás, az USZ-nek
                   kettővel történő növelésére

/KB(1)*P(0)/       B(0—7, 8—15)=0—F(EM), $F(EM)=2
/KJ(3)*P(0)/       J=USZ,
/KOP(0)*P(0)/      C(31)=0,
/KLP(0)*P(0)/      FS=Q, Q=S,
/KFS(3)*P(0)/      USZ←FS,                                            (8.21)
/KST(0)*P(0)/
/KBB(0)*P(1)/      H(6)←1,
/KJB(0)*P(1)/      H(7)←0,
/K*P(1)/           H(0—5)←F(CKC),    $F(CKC)=„xxxxxx"
/G*P(2)/           F←MT(H)
```

Az előző leírás időzítése a következő. A jel a regiszterből a párhuzamos összeadón és a léptetőregiszteren keresztül halad és a fősínt a P(0) órajelfázisban éri el. A fősíntől a regiszterhez történő adatátvitel, valamint a címátvitel a H regiszterbe a P(1) órajelfázisban történik. A mikroutasítást az MT vezérlőtárból a P(2) fázisban hívják le és ezalatt jut el az F regiszterbe.

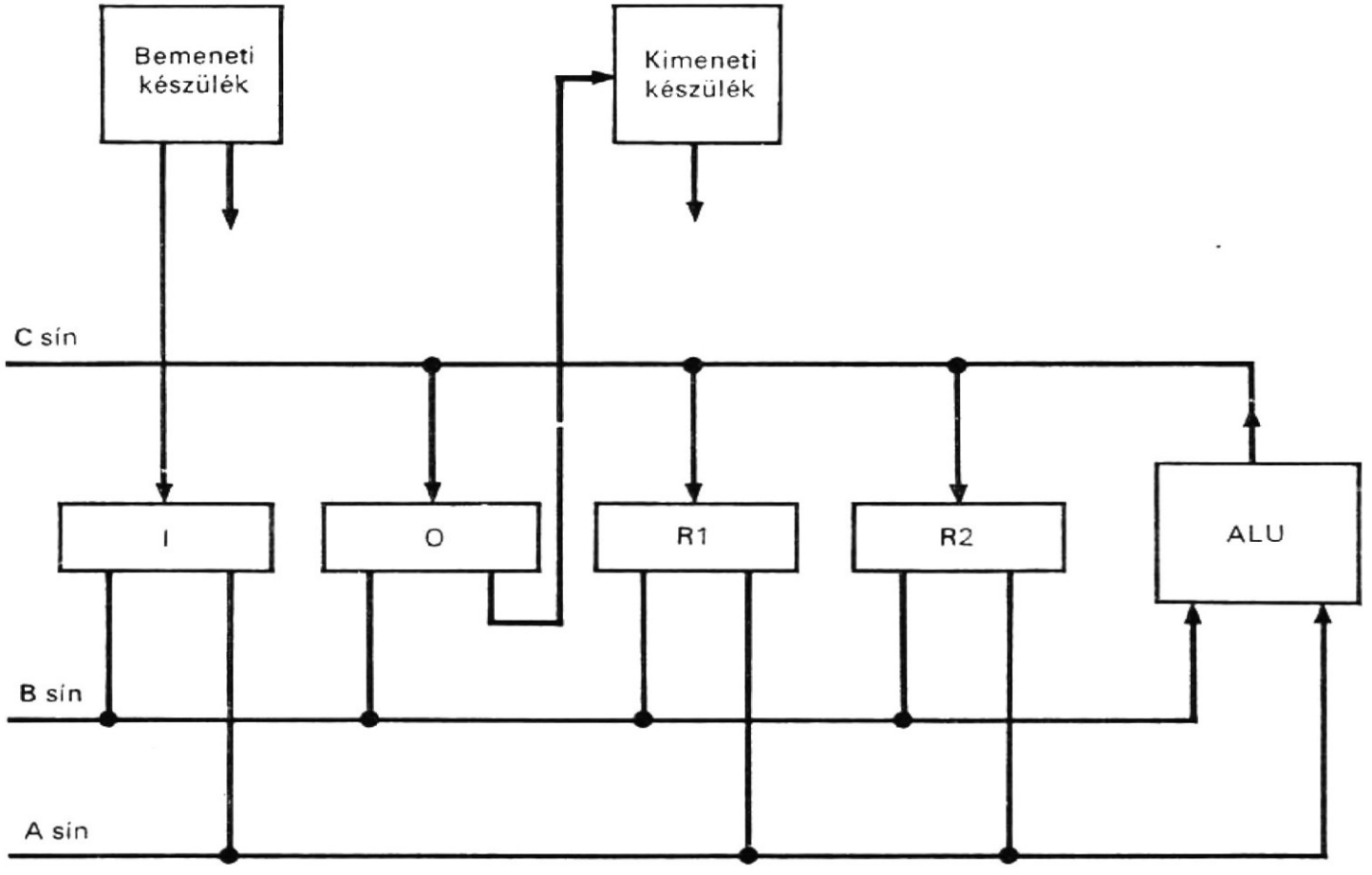


8.8. ábra
Be-, ill. kimeneti adatátvitel

8.2.6.
Mikroprogramozott B/K (I/O) vezérlőegység

A B/K egység nemcsak egy vagy több be-, ill. kimeneti készüléket működtet, hanem a számítógép adatcsatornájával is kommunikál a B/K illesztésen (interface-en) keresztül, amint az a 8.8. ábrán látható. Bár a B/K illesztést szabványosítani lehet egy számítógépcsaládon belül, a B/K vezérlőegységek eltérők, a különböző be-, ill. kimeneti készülékektől függően. Ha a B/K vezérlőegységet mikroprogrammal vezérlik, a mikroprogramozott B/K vezérlőegység, vagy rövidítve MVE (**MI**croprogrammed **C**ontrol **U**nit — MICU) elegendően rugalmas lehet ahhoz, hogy a be-, ill. kimeneti berendezések széles választékához igazodjék. Ebben a fejezetben egy olyan MVE-t írunk le, amelyről McGee és Petersen számolt be [8].

Az MVE négy 8-bites adatregiszterből (R1 és R2 operandusregiszter, I (input) bemeneti, O (output) kimeneti regiszter), három adatsínből (A, B és C) és egy aritmetikai és logikai egységből (ALU) áll (8.9. ábra). Az A és B sín látja el bemenő adatokkal az ALU-t és a C sínre kerül az ALU kimenet. Bármely, az A és B sínhez kapcsolódó



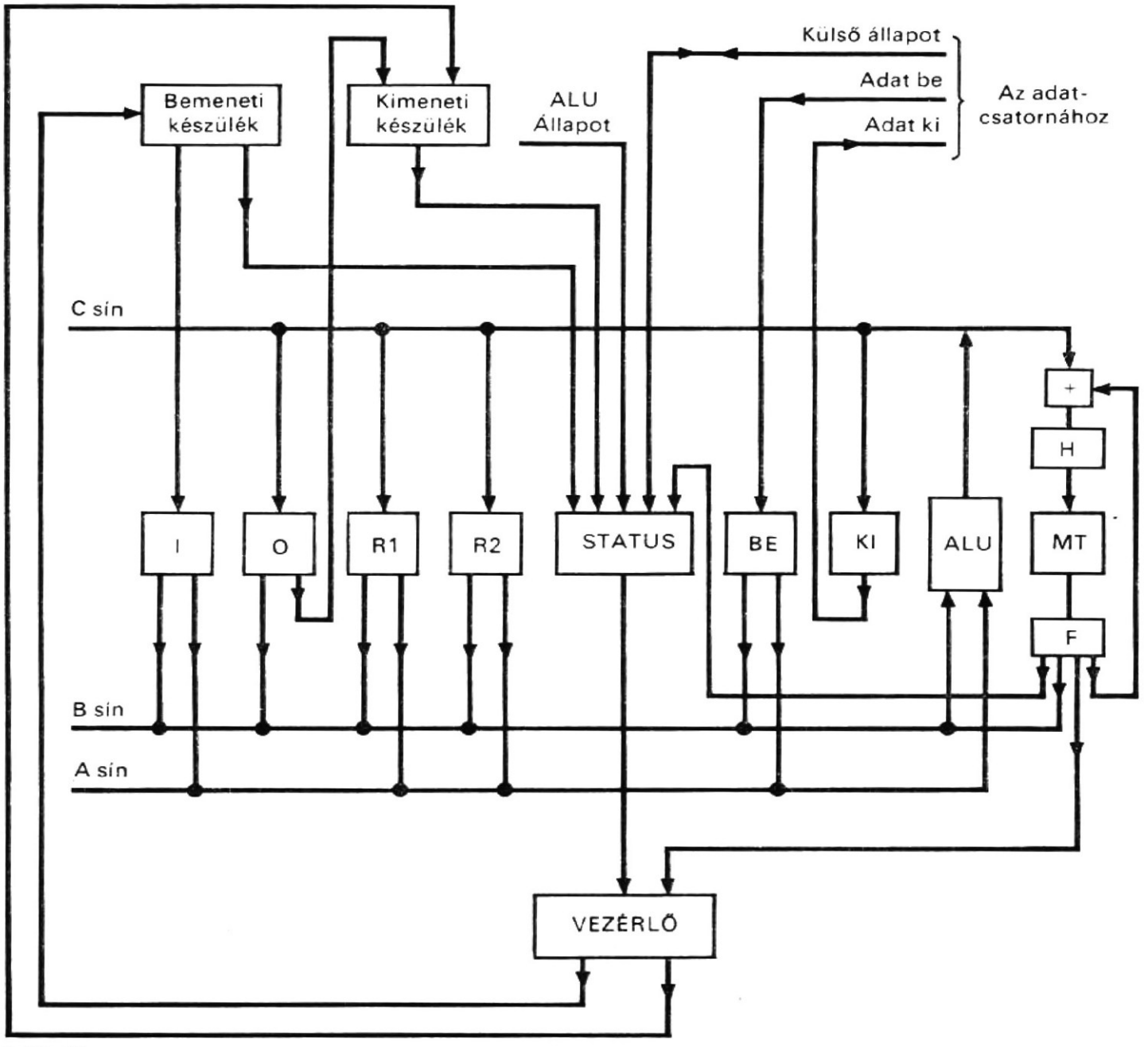
8.9. ábra
A MVE struktúrájának egy része

regiszter adatokat szolgáltathat az ALU-nak és bármelyik, a C sínnek összekapcsolt regiszter adatokat kaphat az ALU-tól. A regiszterek csatlakozhatnak az A, B és C sínhez, kivéve az I regisztert, amely nem kapcsolható a C sínhez és az O regisztert, amely nem kapcsolható az A sínhez. Az ALU egyszerű aritmetikai és logikai műveleteket tud végezni a bemenő adatokkal, pl. összeadást, kivonást, LOGIKAI—ÉS, LOGIKAI—VAGY műveleteket.

Az MVE-nek van másik négy 8-bites regisztere:

— a BE bemenetvezérlő regiszter;
— a KI kimenetvezérlő regiszter;
— a STATUS állapotregiszter;
— a VEZÉRLŐ jelű készülékvezérlő regiszter, amint a 8.10. ábrán látható.

A BE regiszter fogadja az adatokat a csatornából és a KI regiszter továbbítja az adatokat a csatornába. A VEZÉRLŐ regiszter vezérlőjeleket szolgáltat a bemeneti vagy a kimeneti készülékhez. A STATUS regiszter valamennyi bitje valamilyen belső vagy külső állapotot jelez. A STATUS regiszter bitjeit külső forrás állíthatja be vagy állít-



8.10. ábra
A MVE struktúrája

hatja vissza (pl. egy csatorna), hogy jelezze: adat érkezett a regiszterbe. Beállítható egy belső állapotnak, mint pl. valamelyik ALU-állapotnak, vagy a mikroutasításban levő egyik bit állapotának a tárolására is.

Az MVE-nek van egy mikroprogramozott vezérlőrésze, amely hasonlít a 8.2. ábrán bemutatott mikroprogramtárhoz. A mikroprogramozott vezérlőrész az MT mikroprogramtárat, a H a címregisztert és az F mikroutasítás-regisztert tartalmazza. A mikroprogramtár kapacitása 256 — 45-bites — szó. Továbbá nyolc bemeneti adatvonala (D-BE), nyolc kimeneti adatvonala (D-KI) és nyolc külső állapotvonala (K-STA) van. A fentiekben leírt MVE-t az alábbi utasítások adják meg.

Comment, az MVE struktúrája

```
Register,   R1(0—7),                    $operandusregiszter
            R2(0—7),                    $operandusregiszter
            I(0—7),                     $bemeneti regiszter
            O(0—7),                     $kimeneti regiszter
            BE(0—7),                    $bemenetvezérlő regiszter
            KI(0—7),                    $kimenetvezérlő regiszter
            STATUS(0—7),                $állapotregiszter          (8.22)
            VEZÉRLŐ(0—7),               $készülékvezérlő regiszter
            H(0—7),                     $címregiszter
            F(0—44),                    $mikroutasítás-regiszter
Bus,        A, B, C                     $A,B,C sín
Terminal,   D—BE(0—7),                  $adat-be vonalak
            K—KI(0—7),                  $adat-ki vonalak
            K—STA(0—7),                 $külső állapotvonalak
Memory,     MT(H)=MT(0—255, 0—44),      $mikroprogramtár
```

A mikroutasítás 45 bitből áll, amelyeket 10 mezőre osztanak, amint az a 8.11. ábrán látható. A bitek számát és az egyes mezők jelentését a 8.11. táblázat mutatja. Az első három, CN, CL és CJ mező határozza meg a következő mikroutasítás címét. A CN mező tartalmazza a cím hét nagyobb helyiértékű bitjét, a CL mező a feltételkódot, amely a cím kisebb helyiértékű bitjét határozza meg, amint azt az előzőkben leírtuk. A CJ specifikálja, hogy a következő mikroutasítás címét a CN és a CL mezőkből kaptuk-e, vagy valamilyen más elágaztatási forrásból, mint pl. az R1 regiszterből. Ennek a három mezőnek az alkalmazásával minden mikroutasításban specifikálni lehet a következő mikroutasítás címét, az esetleges kétutas elágazással együtt.

8.11. táblázat

A MVE mikroutasítás formátuma

Mező jele	Bitpozíciók	Jelentés
CN	0—6	A KVSZC* nagyobb helyértékű bitjei
CL	7—9	Elágaztatás feltétel-kód a KVSZC*-hez
CJ	10—11	Elágaztatás forrás a KVSZC*-hez
COP	12—15	Műveleti kód az ALU-hoz
CA	16—18	Az A sínhez csatlakoztatható regiszterek vezérlése
CB	19—21	A B sínhez csatlakoztatható regiszterek vezérlése
CC	22—24	A C sínhez csatlakoztatható regiszterek vezérlése
CK	25—32	Emit-mező
CS	33—36	Az állapot-regiszter bit-címe
CX	37—44	Külső kód vagy konstans

* A KVSZC jelentése következő mikroutasítás címe.

CN	CL	CJ	COP	CA	CB	CC	CK	CS	CX
7	3	2	4	3	3	3	8	4	8

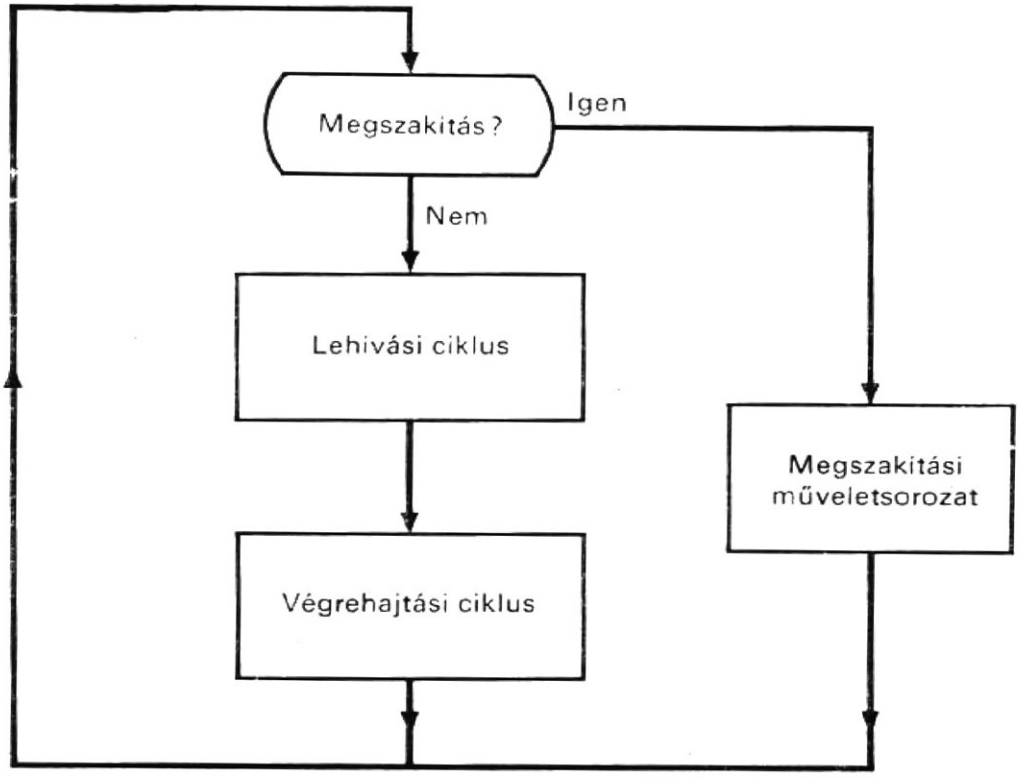
8.11. ábra
A MVE mikroutasításának mezői

A CA, CB, CC, CK és COP mezők az ALU által végzett mikroműveleteket határozzák meg. A CA és CB mező azt a két regisztert jelöli ki, amelyek az ALU-nak szolgáltatnak bemeneti adatokat, míg a CC mező azt a regisztert, amelynek feladata az ALU-ból jövő kimenet fogadása. A CK mező emitmező, amelyet arra használnak, hogy konstanst helyezzen az ALU két bemenetének egyikére. A COP mező az ALU által végzendő műveletet specifikálja. A CS mező 1 vagy 0 értéket ad valamelyik STATUS regiszterbitnek, egy megválasztott feltétel szerint; ez a feltétel a mikroutasítás végrehajtása során léphet fel. A CX mező nyolc vezérlőbitet tartalmaz, amelyeket külső kódként vagy külső konstansként lehet felhasználni. Például, ha a CX mező a 000010000 kódot tartalmazza, a mikroutasítás egy 8-bites bemenő adat fogadását hozhatja létre a BE regiszterben. Röviden, egyetlen mikroutasítás választja ki a következő mikroutasítás címét, hajt végre egy aritmetikai vagy logikai műveletet, állít be egy bitet az állapotregiszterben és küld kódot vagy konstanst külső vonalakra.

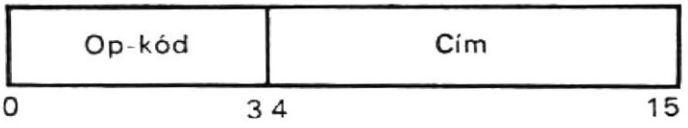
8.3.
A központi vezérlés szervezése

A tárolt programú számítógép működése közben állandóan végrehajtja a vezérlőciklust. A legegyszerűbb vezérlőciklus két ciklusból áll:

— a lehívási ciklusból (vagy utasításlehívó ciklusból);
— a végrehajtás (vagy utasításvégrehajtó) ciklusból.



8.12. ábra
CPU vezérlési ciklusa



8.13. ábra
Utasításformátum

A CPU váltakozva hajtja végre a lehívási és a végrehajtási ciklust. Ahhoz, hogy előre nem látható eseményekre is válaszolni tudjon, a CPU megengedi, hogy a vezérlő ciklust meg lehessen szakítani a végrehajtási ciklus befejezése után (8.12. ábra). Ebben a szakaszban a vezérlőciklust megvalósító központi vezérlés szervezésének kérdéseivel foglalkozunk.

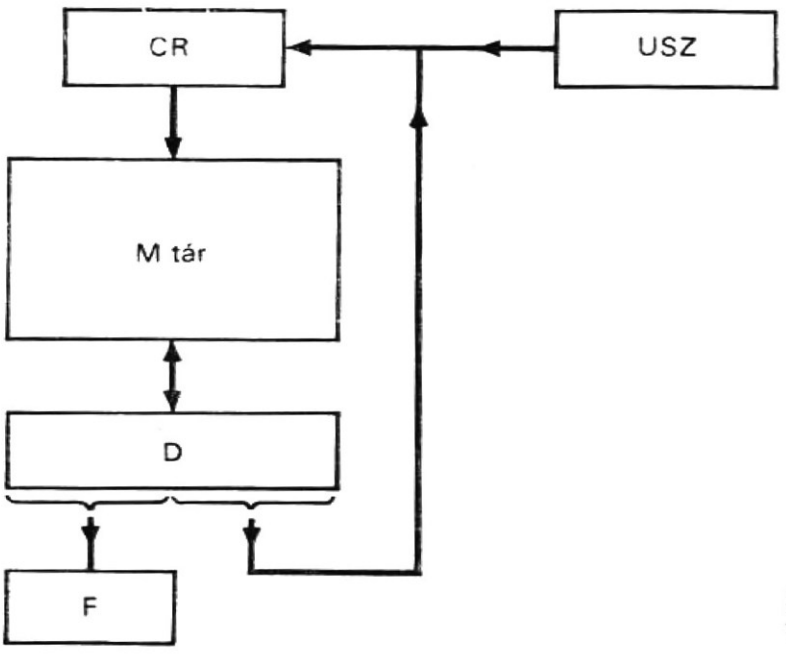
8.3.1. Sorrendbe állítás

A tárolt programú számítógéptől azt kívánjuk, hogy hajtsa végre a főtárban tárolt program utasításait. A műveletsorozat a következő utasítás címének meghatározásával kezdődik. Továbbá a legtöbb jelenlegi számítógéptől (különösen a multiprogramozott számítógépektől) azt is megkívánják, hogy képesek legyenek egyik programból egy másik program sorrendbe állítására (az egyik program felügyelőprogram lehet). Ez gyakran a megszakítás segítségével történik. Röviden a sorrendbe állítást vagy a következő utasítás címének vezérlésével, vagy a számítógép megszakítási lehetőségének felhasználásával valósítják meg.

A következő utasítás címének vezérlésével végzett sorrendbe állítás megoldható:

a) a normál sorrenddel;
b) utasításelágazással;
c) a VÉGREHAJTÁS (EXECUTE) utasítás alkalmazásával.

A sorrendbe állítás kivitelezésének módja függ az utasítás formátumától. A leggyakrabban használt utasításformátum egy műveleti kódot (op-kódot) és egy operanduscímet (vagy egy utasítás címét, elágaztatási utasítás esetében) tartalmaz, amint az a 8.13. ábrán látható. Vizsgáljuk meg a 8.14. ábrán látható egyszerű vezérlési elrendezést, amelyben az előbb említett utasításformátumot alkalmazzuk. A normál sorrend azt a következő utasítást hívja le, amely a pillanatnyi utasítás helyéhez képest követ-



8.14. ábra
Vezérlésszervezés

kező utasítást hívja le, ez a pillanatnyi utasítás helyéhez képest következő legközelebbi címen található. Ezt a következő példán mutatjuk be.

```
Comment, műveletsorrend szervezése                                    (8.23)
Register,   CR(0—14),          $címregiszter
            D(0—35),           $tartalomregiszter (puffer)
            USZ(0—14),         $utasításszámláló
            F(0—8),            $op-kód regiszter
            IR(0—3),           $megszakítás- (interrupt) kérés regiszter
            IRPT,              $megszakításssorrend vezérlő regiszter
Memory,     M(CR)=M(0—32767, 0—35),

Comment, normál műveletsorrend                                        (8.24)
/START/     IF (IR≠0) THEN (GOTO INTERPT);
            CR←USZ;
            D←M(CR), USZ←countup USZ;
            F←D(0—8), CR←D(21—35);
            utasításciklus;
            GOTO START;
/INTERPT/   IRPT←1;
            .....................
            megszakítás sorozat,
            .....................
            GOTO START;
```

A fenti utasítás egy utasításlehívó ciklust ír le. Az IR regisztert először megvizsgálják, van-e egyáltalán megszakításkérelem. Ha van egy vagy több, megindítják a megszakítási műveletsorozatot, miközben az IRPT regiszter 1 értéket kap. Ha nincs, megkezdődik a lehívási sorozat, miközben az US utasításszámlálóban levő következő utasításcímet átviszik a CR címregiszterbe. Ezután kiolvassák a következő utasítást a tárból; egyúttal az utasításszámlálót eggyel megnövelik, hogy ismét a következő utasítás címét tárolja. A most már a D tartalom- (puffer-) regiszterben levő utasítás műveletikód-részét és operanduscím-részét átviszik az F, ill. a CR regiszterekbe. Ezzel az utasításlehívó ciklus be is fejeződött.

Ha a sorrend elágazódik egy elágaztatási utasítással, a következő utasítás címét az elágaztatási utasítás címmezőjéből veszik, nem pedig az utasításszámlálóból. Ezt a következőkben magyarázzuk meg.

```
Comment, műveletsorrend elágaztatási utasítással                      (8.25)
            CR←USZ;
            D←M(CR), USZ←countup USZ;
            F←D(0—8), CR←D(21—35);
            USZ←D(21—35);
```

A fenti példa utasításlehívó ciklus része ugyanaz maradt (kivéve, hogy itt elhagytuk a megszakítást). A lehívási ciklus végén a következő utasítás címe már a címregiszterben van; csak az utasításszámlálót kell beállítani (felfrissíteni).

A VÉGREHAJTÁS (EXECUTE) utasítás arra az utasításra ágazik el, amelynek címét a VÉGREHAJTÁS utasítás címmezőjében levő cím határozza meg, végrehajtja azt, és visszatér az eredeti program következő utasítására. Ezt egy-egy utasításos szubrutinnak tekinthetjük. Erre példa a következő:

```
Comment, műveletsorrend a VÉGREHAJTÁS utasítás esetén                 (8.26)
/LEHÍVÁS/   CR←USZ;
```

```
D←M(CR), USZ←countup USZ; $ VÉGREHAJTÁS utasítás a
                              D-ben
F←D(0—9), CR←D(21—35);
D←M(CR);                      $elágaztatott utasítás D-ben
az elágaztatott utasítás végrehajtása
GOTO LEHÍVÁS;
```

A fenti példákban a sorrendbe állítás a következő utasítás címének vezérlésével történt. A megszakítási lehetőséggel végzett sorrendbe állítást a következőkben írjuk le.

8.3.2. Címzés

A főtárcímzést a 7. fejezetben írtuk le és értelmeztük. Részletesen leírtuk a közvetlen címzést, direkt címzést, indirekt címzést, indexelt címzést, relatív címzést, báziscímzést és regisztercímzést. Ezek a számítógépszervezés rendkívül fontos jellemzői. A 8.15. ábrán a vezérlőciklus folyamatábrája látható, indexeléssel és indirekt címzéssel, a lehívási ciklusban. A következőkben az indexelt és az indirekt címzéses lehívási ciklust megvalósító vezérlés szervezését tárgyaljuk.

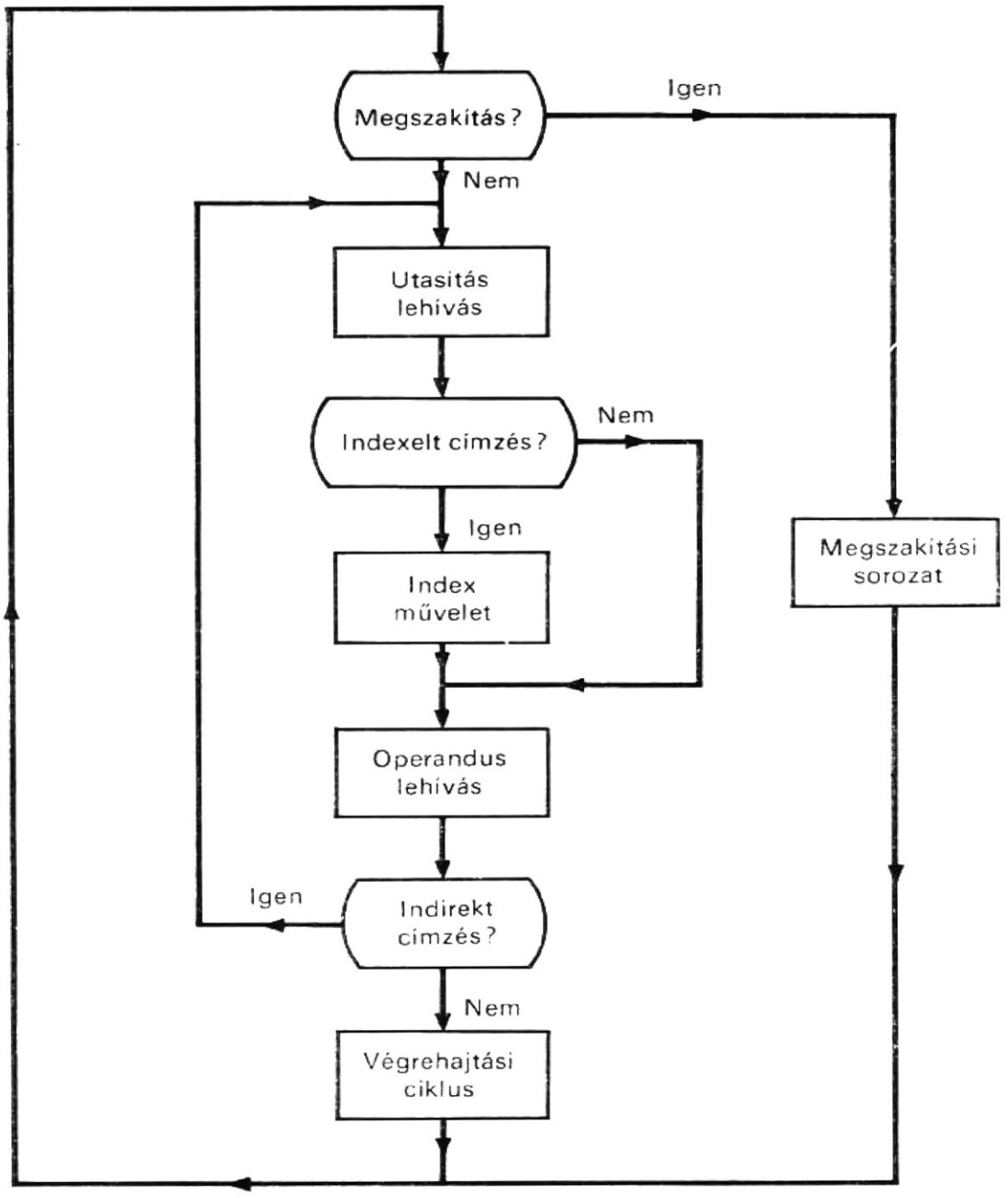
```
Comment, indexelt és indirekt címzéses vezérlés szervezése          (8.27)
Register,          CR(0—14),              $címregiszter
                   D(0—35),               $tartalomregiszter
                   USZ(0—14),             $utasításszámláló
                   I,                     $indirekt címzés jelölőbit (flag)
                   AC(0—35),              $akkumulátor
Array-register,    XR(1—3, 0—14),         $indexregiszterek
Subregister,       D(ADDR)=D(21—35),      $az utasításcímmezője
Memory,            M(CR)=M(0—32677,0 —35),
```

```
Comment, az utasítás lehívása                                       (8.28)
                   CR←USZ;                $a következő utasítás címének
                                            átvitele a CR-be
                   D←M(CR);               $utasítás kiolvasása a tárból
```

```
Comment, indexelés                                                  (8.29)
                   IF (D(18—20)=0) THEN (CR←D(ADDR)),
                   IF (D(18—20)=1) THEN (CR←D(ADDR) add XR(1,)),
                   IF (D(18—20)=2) THEN (CR←D(ADDR) add XR(2,)),
                   IF (D(18—20)=3) THEN (CR←D(ADDR) add XR(3,)),
                   OP←D(0—11),            $az op-kód tárolása
                   I←D(12);               $az indirekt címbit tárolása
                   D←M(CR);               $operandus kiolvasása a tárból
```

```
Comment, indirekt címzés                                            (8.30)
/X/                IF (I=1) THEN (CR←D(ADDR)) ELSE (GOTO Y);
                   IF (I=1) THEN (D←M(CR));
                   IF (I=1) THEN (I←D(12), GOTO X);
/Y/                AC←D;
```

A 7. fejezet dinamikus betöltőjében leírt címzési eljárást *dinamikus címzésnek* nevezhetjük, mivel valamely utasítás vagy operandus tényleges címe a tárban nem isme-



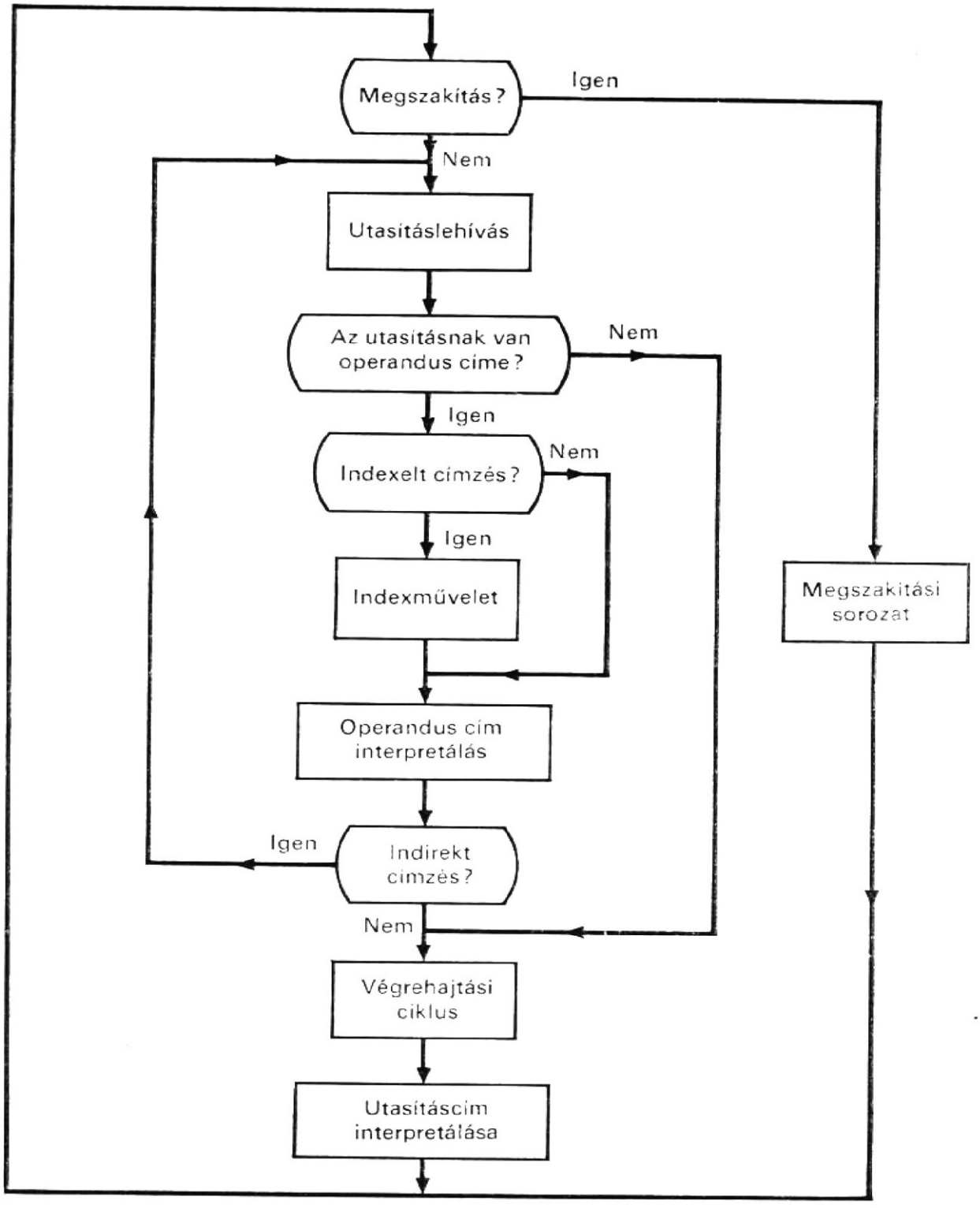
8.15. ábra
CPU vezérlési ciklus, indexelt és indirekt címzéssel

retes a végrehajtás idejéig. A végrehajtás során a tárbeli címet az asszociatív tárban tárolt táblázatból határozzák meg és ezután viszik a tár címregiszterébe, az utasítás vagy az adatszó elérése végett.

Ha dinamikus címzést használunk egy vezérlőciklusban, a vezérlőciklus folyamatábrája a 8.16. ábrán bemutatott vagy ahhoz hasonló lehet. Ezen két új művelet van:

— az operandus címének és
— az utasítás címének interpretálása (meghatározása).

Az operandus címének interpretálása az az eljárás, amellyel megkapjuk az aktuális operanduscímet, az asszociatív tárban levő Z lapcím felhasználásával, amint azt a 7.5.4. pontban az operandus címének lehívása kapcsán leírtuk. Az utasítás címének interpretálása az az eljárás, amellyel megkapjuk az aktuális utasításcímet az asszociatív tárban levő X lapcím felhasználásával, amint azt a 7.5.3. pontban az utasítás sorrendbe állítása kapcsán leírtuk.



8.16. ábra
CPU vezérlési ciklus, dinamikus címzéssel

8.3.3. Prioritásos megszakítás

Amikor egy digitális számítógép működik, előfordulhatnak bizonyos nemkívánt események vagy kivételes állapotok, mint például összeadási túlcsordulás, paritáshiba, szolgálatkérés B/K művelet esetén, vagy real-time bemenetből jövő jel megérkezése. Ezek az események és állapotok viszonylag ritkán fordulnak elő és előre megjósolhatatlanok. Azonban bármikor is következzenek be, a CPU-nak a lehető leggyorsabban

ki kell szolgálnia őket. Első időkben programhurkot használtak egy-egy ilyen esemény ellenőrzésére és várására; ennek eredménye a hardware-kihasználtság csökkenése lett. Napjainkban elterjedten használják a számítógépekben a megszakítást. Amikor megszakítás jön létre, az eseményt részben a hardware, részben a software kezeli. Fontos, hogy a kezelésnek ebben az időszakában a megszakítás igen rövid legyen, hogy a következő megszakítást kis késéssel lehessen ellátni.

A megszakításnak nagy digitális számítógépen való első alkalmazása talán az Univac Scientific 1103A számítógépen történt, az N.A.C.A.-nál (Cleveland), amint arról Mersel [1] 1956-ban beszámolt. Leírása szerint a megszakítás a következőképpen működik. Amikor megszakítási jel jut az Univac Scientific-re, a számítógép a pillanatnyi utasításának befejezése után azonnal megjegyzi azt a helyet, ahol a következőkben az utasítást normálisan folytatta volna, és egy meghatározott, rögzített címre ugrik, ahol az említett utasításcímet tárolni lehet és ahonnan a számítógép információt kap arról, mit is kell tennie, miért történt a megszakítás.

A korszerű digitális számítógépek megszakításrendszere egyre összetettebbé és egyre hatékonyabbá válik a korai gépekéhez képest. Egy vagy néhány megszakítási forrás helyett ma már sok ilyen forrást tud kezelni; ilyen lehet pl. az aritmetikai túlcsordulás, meg nem engedett (illegális) műveleti kód, terminálóktól jövő kapcsolatkérés, hibás gépműködés, az időadótól származó megszakítás, a tárvédelem megsértése, be-, ill. kimeneti művelet befejezése stb. A hasonló megszakítási forrásokat rendszerint azonos osztályokba sorolják; mindegyik osztály kap egy prioritásszintet, fontosságának és sürgősségének megfelelően. A különböző osztályokból érkező szimultán megszakításokat a CPU azok prioritási szintje szerint fogadja; vagyis a legnagyobb prioritásút fogadja először. Kívánatos, hogy ezeket a prioritási szinteket programozással módosítani vagy változtatni lehessen. Általános gyakorlat az, hogy maszkbitet rendelnek minden egyes megszakításhoz. Amikor valamely megszakítás maszkolva van, azt nem veszik figyelembe. Ámbár egy magasabb prioritásszintről jövő interrupt megszakíthat egy, a CPU által éppen feldolgozás alatt levő interruptot, lehetnek rövid időszakaszok, amelyek alatt nem engedélyezik a CPU működés zavarát; ezt egy egybites engedélyező-lezáró (enable-disable) regiszter vezérli.

A prioritásos megszakítási művelet jobb megértése végett bemutatunk egy prioritásos megszakításrendszert. Struktúrája a 8.17. ábrán látható. Ebben hat regiszter van:

— a BJ bemenőjel-regiszter, amely a három prioritás szinten levő három IN interrupt vonaltól fogadja a jeleket;

— a STATUS állapot — (státusz) — regiszter, amely a három megszakítási állapot vonalhalmaztól (SS1, SS2 és SS3) fogadja a jeleket;

— az IM interrupt maszkregiszter, ez maszkolja a BJ regiszter megfelelő bitjeit;

— az IP prioritásos interrupt regiszter, amely a legnagyobb prioritású bejövő megszakítást tárolja;

— a BE bejövő megszakításszint-regiszter, amely a legnagyobb prioritású bejövő megszakítás prioritásszintjét tárolja;

— a J pillanatnyi megszakításszint-regiszter, amely a CPU által éppen feldolgozott pillanatnyi megszakítás prioritásszintjét tárolja.

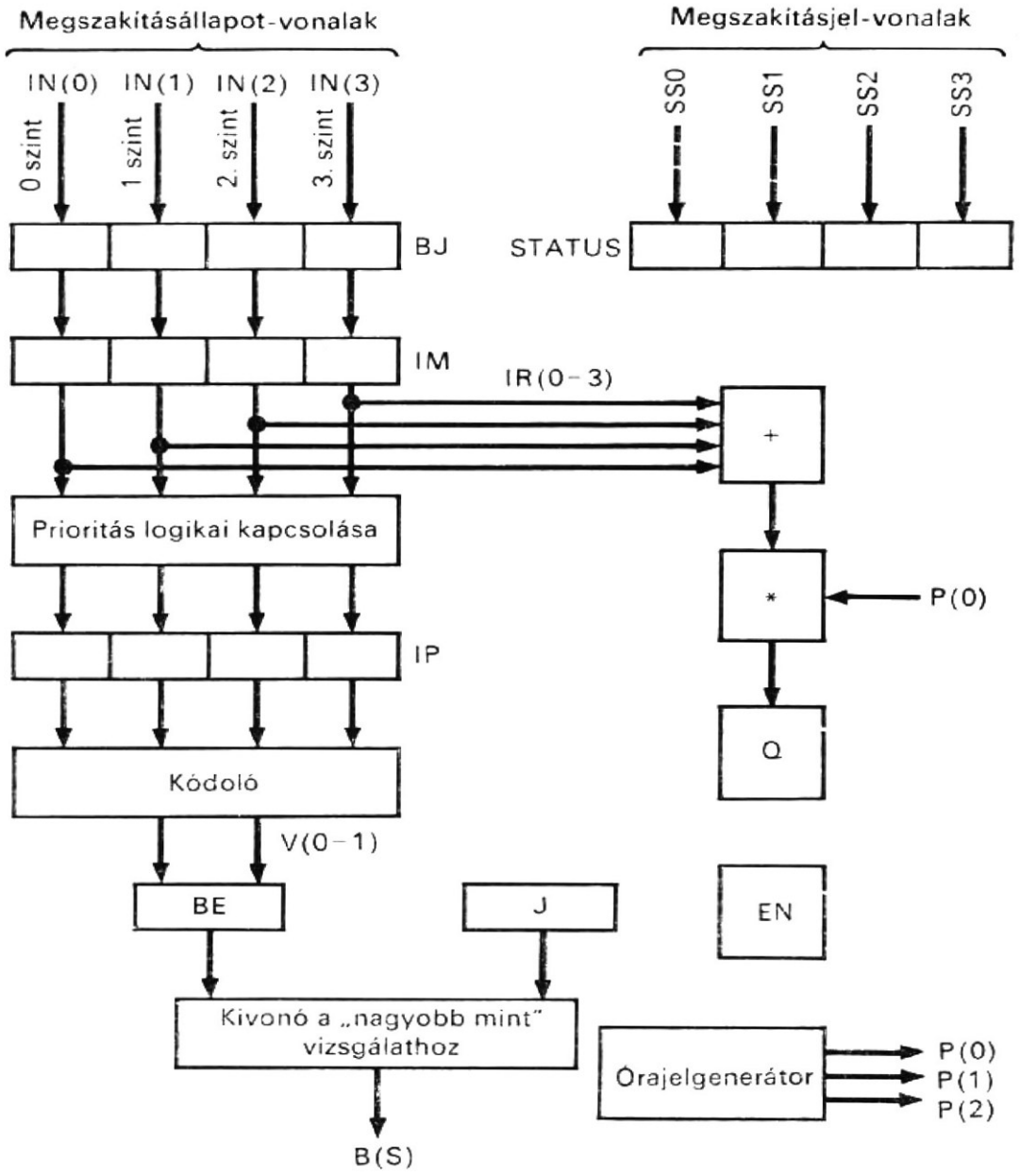
Van még négy egybites regiszter:

— a Q megszakításkérés regiszter, amelynek 1 értéke egy vagy több megszakítási jel előfordulását jelzi;

— az EN megszakításengedélyező regiszter, amelynek 1 vagy 0 értéke engedélyezi, ill. lezárja a *megszakításvizsgáló részsorozatot*;

— az E és I regiszterek, amelyek 1 értéke a végrehajtási, ill. utasításciklust jelzi (a 8.17. ábrán nincs feltüntetve).

Továbbá van egy prioritás logikai kapcsolás, amely kiválogatja a bejövő megszakítások közül a legnagyobb prioritásút, egy kódoló, amely a prioritás szerint kiválasztott megszakítást bináris számmá alakítja át, egy kivonó annak vizsgálatára, hogy a BE re-



8.17. ábra
Prioritásos megszakítás struktúrája

gíszterben levő szám nagyobb-e, mint a J regiszterben levő, valamint egy háromfázisú órajelgenerátor.

A fenti prioritásos megszakítási rendszer struktúráját a következő CDL utasítások irják le.

```
Comment, prioritásos megszakítási rendszer                    (8.31)
Register,   BJ(0—13),      $megszakítási jel regiszter
            IM(0—3),       $megszakítási maszkregiszter (engedélyezve,
                           ha 1)
            IP(0—3),       $prioritásos megszakításregiszter
            BE(0—1),       $bejövő megszakításszint
            J(0—1),        $pillanatnyi megszakításszint
            EN,            $megszakításengedélyezés regiszter (ha 1)
            Q,             $megszakításkérés regiszter (ha 1)
            E,             $végrehajtásiciklus-regiszter (ha 1)
            I,             $utasításciklus-regiszter (ha 1)
            STATUS(0—3),   $a megszakítási forrás állapotregisztere
```

```
Clock,          P(0—2),
Encoder,        V(0—1)=IP,          $a legnagyobb prioritású megszakítás kódolója

Comment, bemenő vonalak                                              (8.32)

Terminal,       IN(1—3),            $megszakítási bemenő vonalak
                SS1(1—3),           $megszakítási állapot vonalak
                SS2(1—3),           $megszakítási állapot vonalak
                SS3(1—3),           $megszakítási állapot vonalak

Comment, logikai hálózat a nemmaszkolt megszakítások azonosítására   (8.33)

Terminal,       IR(1—3)*IM(1—3),

Comment, kivonó annak ellenőrzésére, hogy nagyobb-e a bejövő prioritási szint
                                                                     (8.34)
Terminal,       B(S)=J(0)'*BE(0)+BE(0)*B(0)+B(0)*J(0)',
                B(0)=J(1)'*BE(1),                                    (8.35)
Block, STATUS—IN(STATUS(1—3)←IP(1)*SS1+IP(2)*SS2+IP(3)*SS3),

Comment, a megszakítási jel fogadása                                 (8.36)

/P(2)/          BJ←IN,              $a megszakítási jel belépése

Comment, megszakításvizsgáló részsorozat                             (8.37)

/EN*E*P(0)/     Q←IR(1)+IR(2)+IR(3), $a megszakításkérés belépése
                IF (IR(3)=1) THEN (IP←4) ELSE    $prioritás meghatározás
                  (IF (IR(2)=1) THEN (IP←2) ELSE
                  (IF (IR(1)=1) THEN (IP←1) ELSE (IP←0)))),
/Q*E*P(1)/       BJ←0, BE←V, DO STATUS—IN,
/Q*E*P(2)/       Q←0, IF (B(S)=1) THEN (L←1, EN←0),
```

Az IR terminálok 1 esetén maszkolatlan megszakítást jeleznek. Ha a B(S) terminál 1, azt jelöli, hogy a kódolt prioritásszint a BE regiszterben nagyobb, mint a J regiszterben levő prioritásszint. A STATS—IN blokkutasításban a STATUS mikroművelet kiválaszt egyet a három megszakítás státuszvonal-készletből, az IP regiszterben levő értéknek megfelelően (azaz a legnagyobb prioritásszintű megszakítást).

A (8.36) utasítás azokat a bejövő jeleket írja le, amelyeket — ha fellépnek — a P(2) órajelfázisban fogadnak el. A (8.37) utasítás azt mutatja, hogy a megszakításvizsgálati részsorozatot minden végrehajtási ciklusban elvégzik, hacsak nincs gátolva azáltal, hogy az EN regiszter értékét 0-ra állítják vissza. Az első órajelfázisban a megszakítási jel vagy jelek előfordulását a Q regiszter jelzi és ugyanakkor a legnagyobb prioritású megszakítási jelet kiválasztják a prioritás logikai hálózattal, majd ez a jel belép az IP regiszterbe. A második fázisban a BJ regisztert visszaállítják 0-ra és ily módon az készen áll a bejövő megszakítási jelek fogadására a következő órajelfázisban; a kiválasztott megszakítás kódolt prioritásszintje bejut a BE regiszterbe, és végrehajtják a STATUS—IN mikroműveletet, a kiválasztott megszakítás státuszának kiválasztására és fogadására. A harmadik órajelfázisban a Q regisztert visszaállítják 0-ra; megindul a megszakítási sorozat (az L regiszternek 1 értékre állításával). A megszakításvizsgáló részsorozat gátolva van, ha az IP regiszter nem tartalmaz 0-t és ha a B(S) terminál 1-et jelez.

Röviden az egyes végrehajtási ciklusok végén megindul a megszakítási sorozat, ha a megszakítás nem volt gátolva és, ha a kiválasztott megszakítási jelnek magasabb a prioritása, mint a pillanatnyinak.

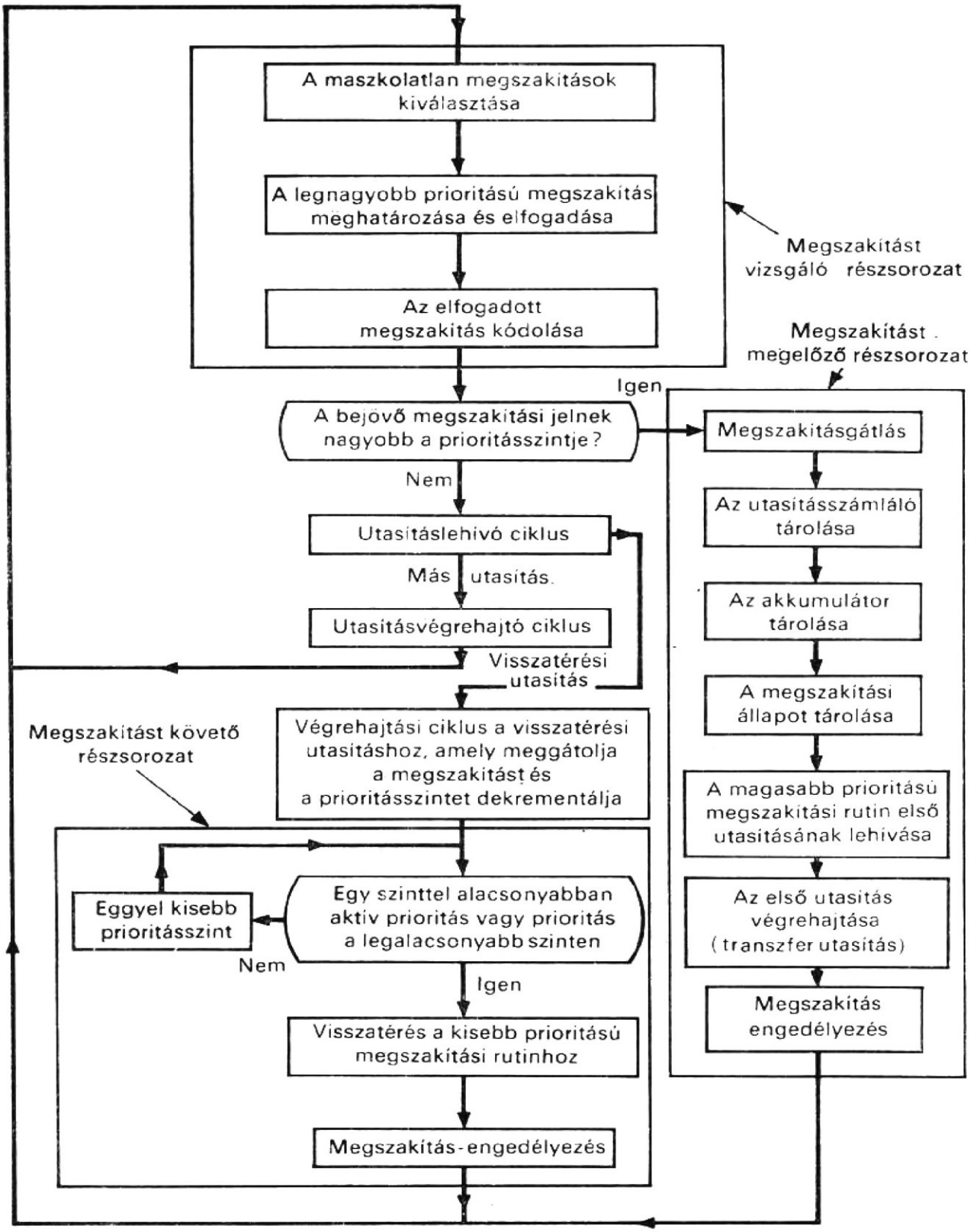
8.12. táblázat
Főtártérkép

Cím	Főtártérkép
0	Utasításszámláló a felhasználói program megszakításkor
1	Utasításszámláló a felügyelőprogram- (supervisor) megszakításkor
2	Utasításszámláló a B/K megszakításkor
3	Utasításszámláló a géphiba-megszakításkor
4	Akkumulátor a felhasználói program megszakításkor
5	Akkumulátor a felügyelőprogram-megszakításkor
6	Akkumulátor a B/K megszakításkor
7	Akkumulátor a géphiba-megszakításkor
8	
9	Állapot (státusz) a felügyelőprogram-megszakításkor
10	Állapot a B/K megszakításkor
11	Állapot a géphiba-megszakításkor
12	
13	Cím a felügyelőprogram-megszakítási rutinkor
14	Cím a B/K megszakítási rutinkor
15	Cím a géphiba-megszakítási rutinkor

8.3.4. Megszakítási műveletsorozat

Az előbbi prioritásos megszakításszervezés szerint egy-egy prioritásszintet kap a három megszakítási vonal mindegyike. Legyen a felügyelő- (supervisor) program, a B/K kérések és a géphibák a megszakítási okok három osztálya, és prioritási szintjeik legyenek rendre 1, 2 és 3. Jelölje a felhasználói programot a 0 prioritásszint. Amikor egy bejövő megszakítást fogadnak, a főtár első négy helyének egyikén tárolják a pillanatnyi programhoz tartozó, az utasításszámlálóban levő tárcímet, amint az a 8.12. táblázatban látható. A CPU regiszterek tartalmát ugyancsak tárolni kell. Ezek a regiszterek: az akkumulátor, az MQ regiszter, az indexregiszterek, az általános célú regiszterek és további regiszterek. Az egyszerűség kedvéért tegyük fel, hogy csak az akkumulátort kell tárolni. Ezt a főtár második négy rögzített helye közül az egyiken tárolják (l. 8.12. táblázat). A három prioritásszint megszakítási állapotát a főtár 8., 9. és 10. helye tárolja. A három megszakítási rutin első utasításait a főtár 13., 14. és 15. helyei tárolják, ami ugyancsak a 8.12. táblázatban látható. Ezek a rutinok képezik a megszakításokat kezelő software-részt.

A vezérlőciklus folyamatábrája a 8.18., ahol a megszakítási sorozatot részletesebben mutatjuk be, mint a 8.13. és 8.15. ábrákon, A megszakítási sorozat a vizsgálati részsorozatból, a megelőző részsorozatból és a követő részsorozatból áll. Az ellenőrzési részsorozatot a (8.37) utasítások írták le: ez a részsorozat ellenőrzi a bejövő megszakítási jeleket az egyes végrehajtási ciklusokban. Az ellenőrzési részsorozat viszont gátolt állapotban van a lehívási ciklus folyamán, és a megszakítást megelőző és azt követő részsorozatok alatt. A megszakítást megelőző részsorozatot azután indítják, mi-



8.18. ábra
A vezérlési ciklus folyamatábrája

után megállapították, hogy a bejövő megszakításnak nagyobb a prioritási szintje, mint azé, amelyet a CPU éppen feldolgoz. Ez a részsorozat gátolja a megszakítást ellenőrző részsorozatot. Tárolja az utasításszámlálót, az akkumulátort és a megszakítási státuszt, majd átadja a vezérlést a legnagyobb prioritású megszakítási rutinnak. Ezt a rutint a főtár kijelölt területén levő cím határozza meg (l. 8.12. táblázatot); ugyanakkor engedélyezést kap a megszakítást ellenőrző részsorozat. Legkorábban a CPU csak akkor fogad egy másik megszakítást, amikor az átvitel megszakítási rutinhoz befejeződött.

Ekkor az egyik megszakítási rutin kapja meg a vezérlést. Ez a megszakítási státusz információ felhasználásával végrehajtja azokat a funkciókat, amelyeket beprogramoztak valamely adott megszakítási forrás kezelésére. A megszakítási rutin utolsó utasítása a visszatérési (return) utasítás kell, hogy legyen. A visszatérési utasítás gátolja a megszakítást vizsgáló részsorozatot és indítja a megszakítást követő részsorozatot. A következő részsorozat megvizsgálja, hogy aktív-e a következő, eggyel alacsonyabb prioritású megszakítás. Ha igen, akkor ezt a kisebb prioritásszintű megszakítást fogja kezelni. Ha nem aktív, akkor folytatódik a vizsgálat, egészen addig, amíg egy alacsonyabb vagy a legalacsonyabb aktív szintet (azaz a felhasználói programot) el nem éri. Az utasításszámlálót és az akkumulátort a felsorolt esetek mindegyikében újból visszaállítják.

A továbbiakban leírjuk a megszakítási sorozatot realizáló műveletsorozatot, amely a 8.18. ábrán látható. Ez felhasználja a (8.31) utasításokban leírt prioritásos megszakítási rendszert és a 8.14. ábrán látható vezérlési struktúrát, amihez az L helyi vezérlés regisztere és az ahhoz tartozó dekódoló járul. Ezt a rendszert a következő utasítások írják le.

```
Comment, megszakítási műveletsorozat                                    (8.38)
Comment, központi vezérlés struktúrája
Register,   CR(0—14),          $címregiszter
            D(0—35),           $tartalom- (puffer-) regiszter
            USZ(0—14),         $utasításszámláló
            F(0—8),            $műveletikód-regiszter
            L(0—2),            $helyi vezérlés regiszter
            AC(0—35),          $akkumulátor
            I,                 $lehívási ciklus, ha 1
            E,                 $végrehajtási ciklus, ha 1
Memory,     M(CR)=M(0—32767,0—35),
Decoder,    K(1—7)=L,

Comment, itt kezdődik a megelőző részsorozat
Comment, folytatás a (8.37) utasításoktól
Comment, az utasításszámláló tárolása                                   (8.39)
/K(1)*P(0)/   CR(0—12)←0,CR(13—14)←J,
/K(1)*P(1)/   D(0)←1, D(1—20)←0, D(21—35)←USZ,
/K(1)*P(2)/   M(CR)←D, L←countup L,

Comment, az akkumulátor tárolása                                        (8.40)
/K(2)*P(0)/   CR(0—12)←1, CR(13—14)←J,
/K(2)*P(1)/   D←AC,
/K(2)*P(2)/   M(CR)←D, L←countup L,

Comment, a megszakítási státusz tárolása                                (8.41)
/K(3)*P(0)/   CR(0—12)←2, CR(13—14)←BE,
/K(3)*P(1)/   D(0—27)←0, D(28—35)←STATUS,
/K(3)*P(2)/   M(CR)←D, L←countup L,

Comment, átmenet a nagyobb prioritású megszakítási rutinhoz             (8.42)
/K(4)*P(0)/   J←BE, CR(0—12)←3, CR(13—14)←BE,
/K(4)*P(1)/   D←M(CR),
/K(4)*P(2)/   F←„TRA", CR←D(ADDR), E←1, EN←1, L←0,
```

```
Comment, végrehajtási ciklus (transzfer utasítás)                    (8.43)
/E*TRA*P(0)/  CR←USZ,
/E*TRA*P(1)/
/E*TRA*P(2)/  IF (Q=0) THEN (I←1), E←0

Comment, itt van a megelőző részsorozat vége
Comment, lehívási ciklus                                             (8.44)
/I*P(0)/      CR←USZ,
/I*P(1)/      D←M(CR), USZ←countup USZ,
/I*P(2)/      F←D(0—8), I←0, E←1,

Comment, a CPU végrehajt egy megszakítási rutint, a nagyobb prioritás-
  szintű megszakítást engedélyezve                                   (8.45)
                  .....

Comment, végül, a CPU visszatérési utasítást hajt végre
Comment, itt kezdődik a következő részsorozat                        (8.46)
/E*RTN*P(2)/  IF (J≠0) THEN (J←countdn J), EN←0, L←5, E←0,

Comment, vizsgálat lehet-e egy kisebb prioritású megszakítási rutinhoz
  visszatérni                                                        (8.47)
/K(5)*P(0)/   CR(0—12)←0, CR(13—14)←J,
/K(5)*P(1)/   D←M(CR),
/K(5)*P(2)/   IF (D(0)=1)+(J=0)) THEN (D(0)←0, L←6)
                                ELSE (J←countdn J),

Comment, visszatérés egy kisebb prioritású megszakítási rutinhoz     (8.48)
Comment, az utasításszámláló visszaállítása
/K(6)*P(0)/   CR(0—12)←0, CR(13—14)←J,
/K(6)*P(1)/   D←M(CR),
/K(6)*P(2)/   USZ←D(21—35), L←countup L, EN←1, E←1,

Comment, az akkumulátor visszaállítása
/K(7)*P(0)/   CR(0—12)←1, CR(13—14)←J,
/K(7)*P(1)/   D←M(CR),
/K(7)*P(2)/   AC←D, IF (Q=0) THEN (I←1, E←0),
                                IF (B(S)=0) THEN (L←0),
Comment, itt végződik a következő részsorozat
```

A fenti leírásban a megelőző részsorozatot a (8.39)...(8.43) utasítások írják le. Ez öt óraperiódust igényel. A D(0) bitnek 1 értéket adó mikroművelet — amelyet a (8.39) utasítások írnak le — feladata annak jelzése, hogy egy adott megszakítási szint éppen aktív; ezt a bitet vizsgálják meg a követő részsorozatban végzett vizsgálatkor [ezt a (8.47) utasítások írják le]. Azokat a helyeket, ahol az utasításszámlálót és az akkumulátort tárolják, a J regiszterből kapjuk meg, amint az a (8.39) és (8.40) utasításokban látható. Azokat a helyeket, ahol a megszakítási státuszt és a nagyobb prioritású megszakítási rutin első utasítását tárolják, a BE regiszterből kapják meg, amint az a (8.41) és (8.42) utasításokból látható. A TRA terminál a transzfer utasítás parancsa.

A megelőző részsorozatot a megszakítási rutin végrehajtása követi. E végrehajtás alatt megengedett egy nagyobb prioritású megszakítás általi interrupt, mivel a megszakítást vizsgáló részsorozat minden lehívási ciklus végén engedélyezve van. Ily módon a következő végrehajtási ciklus és a megszakításvizsgálat szimultán történik a következő órajelperiódus alatt, és a magasabb prioritásszinten levő megszakításra

a vizsgálatot ez után, a végrehajtási ciklus végén végzik el. A megszakítási rutin utolsó utasítása egy visszatérési utasítás kell, hogy legyen; ennek végrehajtása során gátolva van a megszakítást vizsgáló részsorozat. A J regiszter vagy 0 marad, ha 0 volt, vagy 1-gyel dekrementálják, ha nem volt 0, és megkezdődik a következő részsorozat.

A követő részsorozatot a (8.47) és (8.48) utasításokban írtuk le. Funkciói közé tartozik, hogy kikeressen egy kisebb prioritásszintű aktív megszakítást a főtár első négy helyéről és azután adja át a kiválasztott szint megszakításrutinjának. Figyeljük meg, hogy a (8.47) utasítások hurkot alkotnak mindaddig, amíg a D(0) bit nem lesz 1 és a J nem lesz 0 és az RTN végérték nem lesz a visszatérési utasítás parancsa.

8.4. Aszinkron vezérlésszervezés

A korszerű digitális számítógép egy sor rendszertechnikai egységből tevődik össze. Illesztőegységek (interface) szolgálnak az egységek közötti vezérlésre és adatátvitelre. Minden rendszertechnikai egységnek megvan a saját vezérlése, és megvannak a saját, más egységektől független műveletei. Az ilyen szervezési elvet *aszinkronnak* nevezzük.

A rendszertechnikai egységek közötti aszinkron működés illusztrálásaként a következőkben bemutatunk egy egyszerű tárolt programú számítógépet. Ez a számítógép három rendszertechnikai egységből áll:

— központi feldolgozóegységből, a KFE-ből (**C**entral **P**rocessing **U**nit — CPU),
— tárolóegységből, a TE-ből (**M**emory **U**nit — MU) és
— a beviteli-kiviteli egységből, a BKE-ből (**I**nput-**O**utput **U**nit — IOU).

A főtár szóhossza 16 bit. A számokat előjeles 2-es komplemensben ábrázolják. Az utasításoknak a szóban két mezője van: a 4-bites műveletikód-(op-kód) mező és a 12-bites címmező. Az utasításformátum a 8.12. ábrán látható. Nyolc utasítás van: ADD, SUB, LOAD, STORE, JUMP, JUMP-ON-MINUS, INPUT és OUTPUT. A 8.13. táblázat tartalmazza ezen utasítások műveleti kódjait.

8.13. táblázat

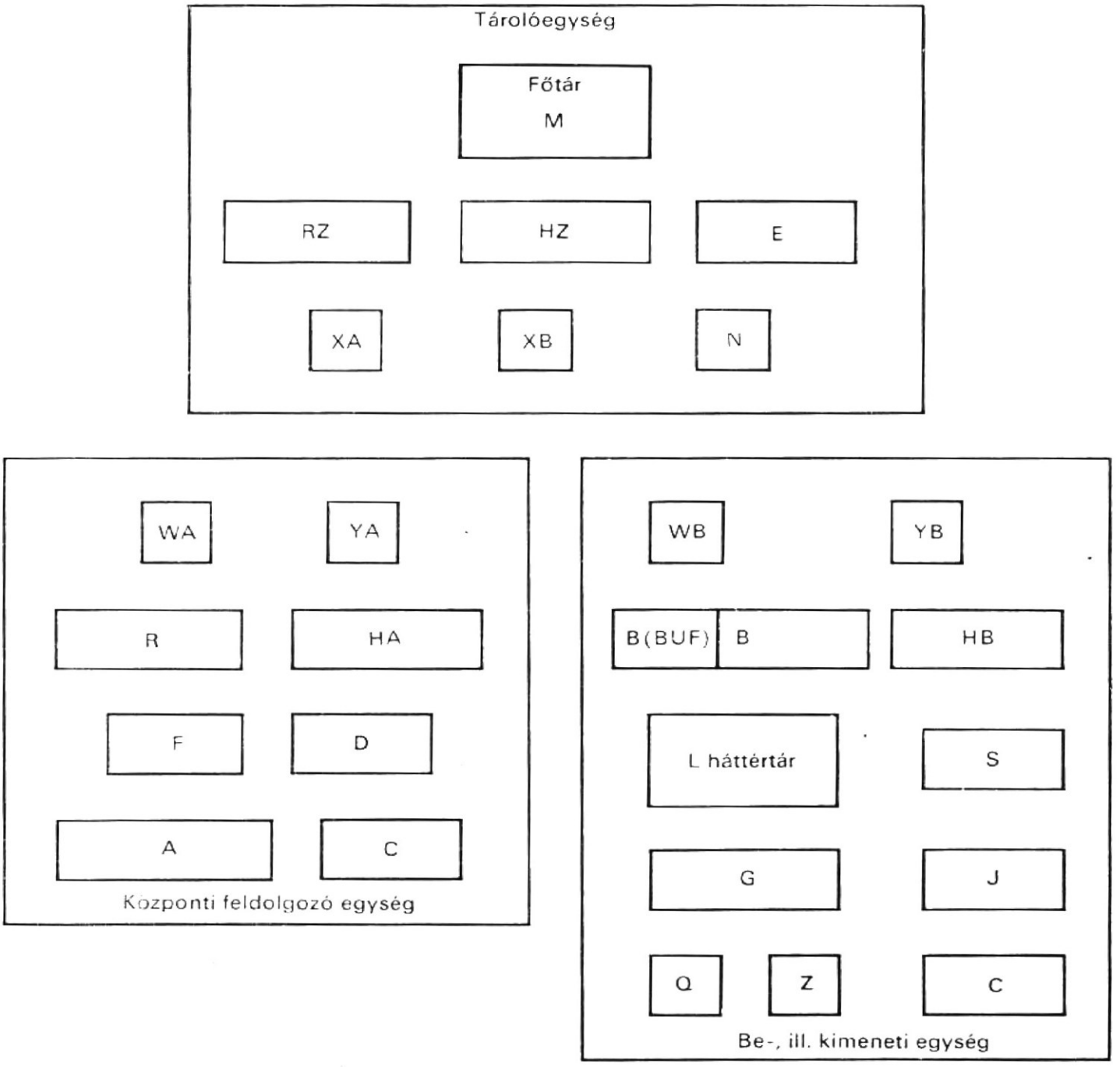
Utasításkészlet

Utasítás	Op-kód*	Szimbolikus kód
Összeadás	06	ADD
Kivonás	05	SUB
Betöltés	02	LDA
Tárolás	01	STO
Ugrás	12	JMP
Ugrás mínusz esetén	11	JOM
Bevitel	15	IN
Kivitel	16	OUT

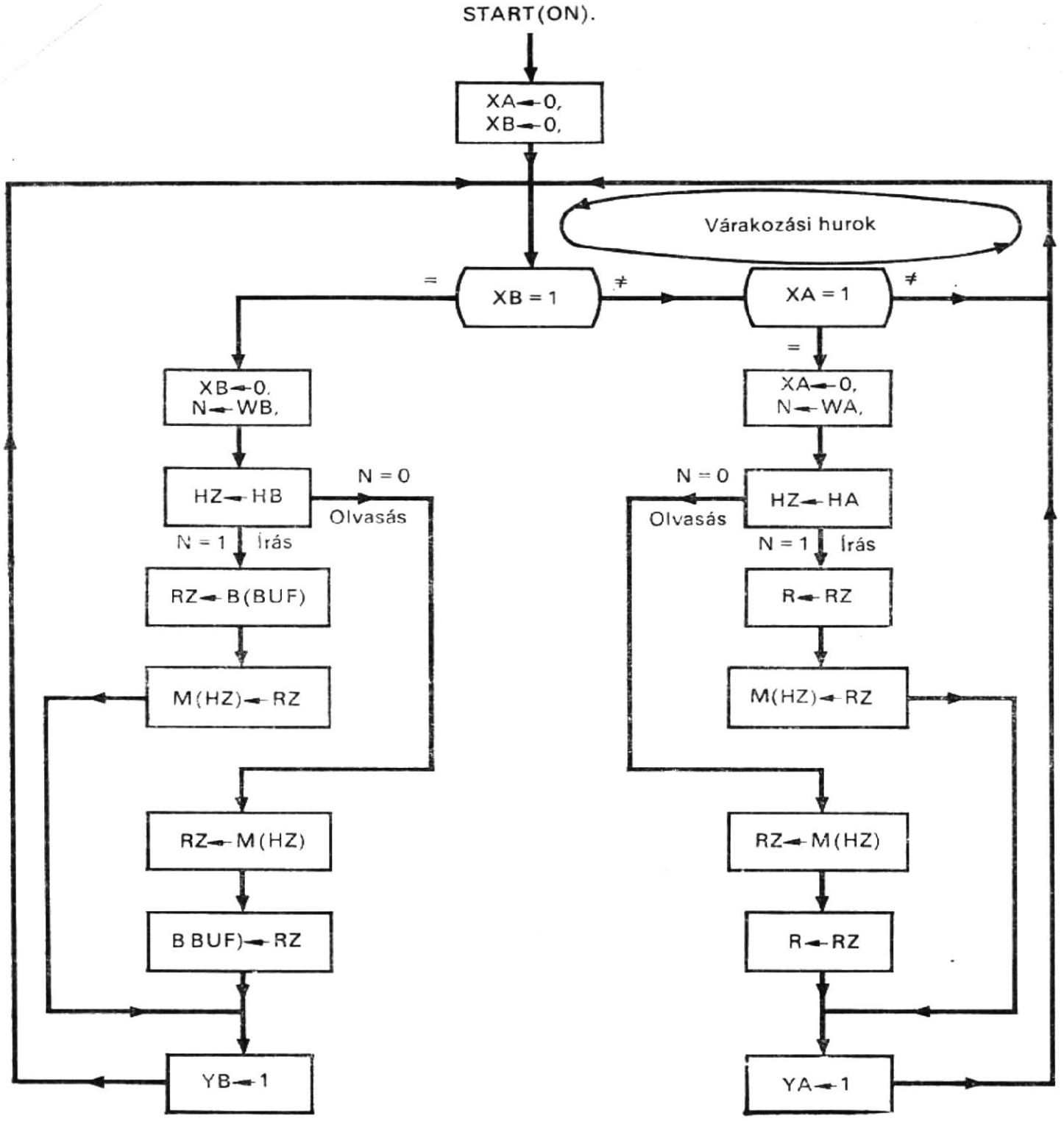
* Oktális.

8.4.1. Tárolóegység

A tárolóegység a 4096 darab 16-bites szavakból álló főtárat, az RZ átmeneti tárolást végző (puffer-) regisztert, a HZ címregisztert és az E tárvezérlőregisztert tartalmazza. Továbbá van három egybites regiszter: az N, az XA és az XB. Az N regiszter olvasási műveletet vezérel, ha értéke 0; egyébként pedig írási műveletre utal. Az XA regiszter a CPU-tól érkező lekérdező jel az M főtár számára, ha az értéke 1. Ha az XB regiszter értéke 1, akkor az BKE-től (IOU-tól) érkező lekérdező jelet tárolja az M főtár számára. Egyszerűség kedvéért feltételezzük, hogy a főtár ciklusideje megegyezik az órajelgenerátor ciklusidejével. (Azt az esetet, amikor ezek nem egyformák, a feladatokban vizsgáljuk meg.) Az alábbiakban leírjuk a tárolóegységnek, a TE-nek ezen regisztereit és a többi funkcionális elemet.



8.19. ábra
A tárolóegység, a központi feldolgozóegység és a be-, ill. kiviteli egység regiszterei



8.20. ábra
A tárolóegység sorrendi működésének folyamatábrája

```
Comment, a TE(MU) struktúrája                                    (8.49)
Register,    RZ(0—15),     $tartalomregiszter
             HZ(0—11),     $címregiszter
             E(0—3),       $tár vezérlőregisztere
             N,            $olvasási művelet, ha 0; egyébként írási művelet
             XA,           $CPU tárlekérdezés, ha 1
             XB,           $IOU tárlekérdezés, ha 1
Decoder,     KE(0—15)=E,
```

```
Switch,     START(ON),
Memory,     M(HZ)=M(0—4095, 0—15),
Clock,      P,
```

A tárolóegységben levő regisztereket a két másik egységben szereplő regiszterekkel együtt a 8.19. ábrán láthatjuk.

A MU sorrendi működését a 8.20. ábrán mutatjuk be. Ezen két olyan műveletsorozat szerepel, amelyeket az XA és XB regiszterek vezérelnek. Amikor az MU működni kezd, ez a két regiszter 0 értéket kap. Először megvizsgálják az XB regisztert; ha az 1, azt jelzi, hogy az IOU kezdeményezi a műveletsorozatot. Ha ez 0, megvizsgálják az XA regisztert. Ha az XA regiszter 1-et tartalmaz, ez a CPU által kezdeményezett műveletsorozatot jelöl. Ha az XA regiszter tartalma 0, ismét megvizsgálják az XB regisztert és várakozási hurok jön létre. Az a tény, hogy az XB regisztert vizsgálják meg először és csak azután az XA regisztert azt jelenti, hogy nagyobb prioritást rendelnek az IOU által kért műveletsorozathoz, mint a CPU kéréséhez.

Az IOU által kért műveletsorozat a 8.20. ábra bal oldalán látható. Ez először nyugtázza a kérést oly módon, hogy az XB regisztert visszaállítja 0-ra. Az IOU-ban levő WB regiszter tartalmát átviszi az N regiszterbe, és a HB regiszter tartalmát a HZ címregiszterbe. Ha az N regiszter 1-et tartalmaz, tároló-írási műveletet kell végezni. Ebben az esetben átviszik az IOU-ban levő B (BUF) részregiszter tartalmát az RZ tartalomregiszterbe, majd ezután az RZ tartalomregiszter tartalmát tárolják az M tárban. Ha az N regiszter 0-t tartalmaz, tárolvasási művelet következik. Ebben az esetben az M tárból egy szót olvasnak ki az RZ tartalom- (puffer-) regiszterbe és az RZ regiszter tartalmát viszik át ezután az IOU-ban levő B (BUF) részregiszterbe. Tekintet nélkül arra, hogy olvasási vagy írási műveletről van-e szó, az IOU-ban levő YB regiszter most 1 értéket kap, és ez informálja az IOU-t arról, befejeződött a kívánt tárművelet. Az IOU visszatér a várakozási hurokba.

A CPU által kért műveletsorozat a 8.20. ábra jobb oldalán látható. Először nyugtázza a kérést, az XA-nak 0 értéket adva és átviszi a CPU-ban levő WA és HA regiszterek tartalmát az N, ill. a HZ regiszterekbe. A művelet ismét csak olvasás, ha az N regiszter 0; egyébként pedig írás. Mindkét esetben hasonló az olvasás vagy írás művelet vezérlése ahhoz, ami az IOU által kért műveletsorozat esetén zajlik le. Most a CPU-ban levő YA regiszter kap 1 értéket, és így értesíti az IOU-t, hogy a kívánt tárművelet befejeződött. Ezután az IOU ismét visszatér a várakozási hurokba.

A 8.20. ábrán bemutatott MU műveletet most CDL utasításokkal is leírjuk.

```
Comment, az MU művelet. Az IOU-nak nagyobb a prioritása, mint a
  CPU-nak                                                    (8.50)
/START(ON)/  XA←0, XB←0, E←0,
/KE(0)*P/    IF (XB=1) THEN (E←1) ELSE (E←8),

Comment, IOU által kért sorozat

/KE(1)*P/    XB←0, N←WB, E←3,
/KE(3)*P/    HZ←HB, IF (N=0) THEN (E←7) ELSE (E←2),
/KE(2)*P/    RZ←B(BUF), E←6,
/KE(6)*P/    M(HZ)←RZ, E←4,
/KE(7)*P/    RZ←M(HZ), E←5,
/KE(5)*P/    B(BUF)←RZ, E←4,
/KE(4)*P/    YB←1, E←0,

Comment, CPU által kért sorozat

/KE(8)*P/    IF (XA=0) THEN (E←0), ELSE (E←9),
/KE(9)*P/    XA←0, N←WA, E←11,
```

```
/KE(11)*P/   HZ←HA, IF (N=0) THEN (E←15), ELSE (E←10),
/KE(10)*P/   RZ←R,E←14,
/KE(14)*P/   M(HZ)←RZ, E←12,
/KE(15)*P/   RZ←M(HZ), E←13,
/KE(13)*P/   R←RZ, E←12,
/KE(12)*P/   YA←1, E←0,
             END
```

8.4.2. Be-, ill. kiviteli egység

Az MU-ban levő főtár 64 blokkra van osztva, amelyek közül mindegyik 64 darab 16-bites szót tartalmaz. Az egyes blokkokat *lapnak* nevezik, tehát az M tárnak 64 lapja van. Az IOU-nak van egy L háttértára, amely 1024 1024-bites szót tartalmaz. Az egyes 1024-bites szavak mérete megegyezik egy lap méretével. Az M főtár és az L háttértár közötti átvitel: egy lap egyidőben. A lapátvitelhez két cím szükséges:

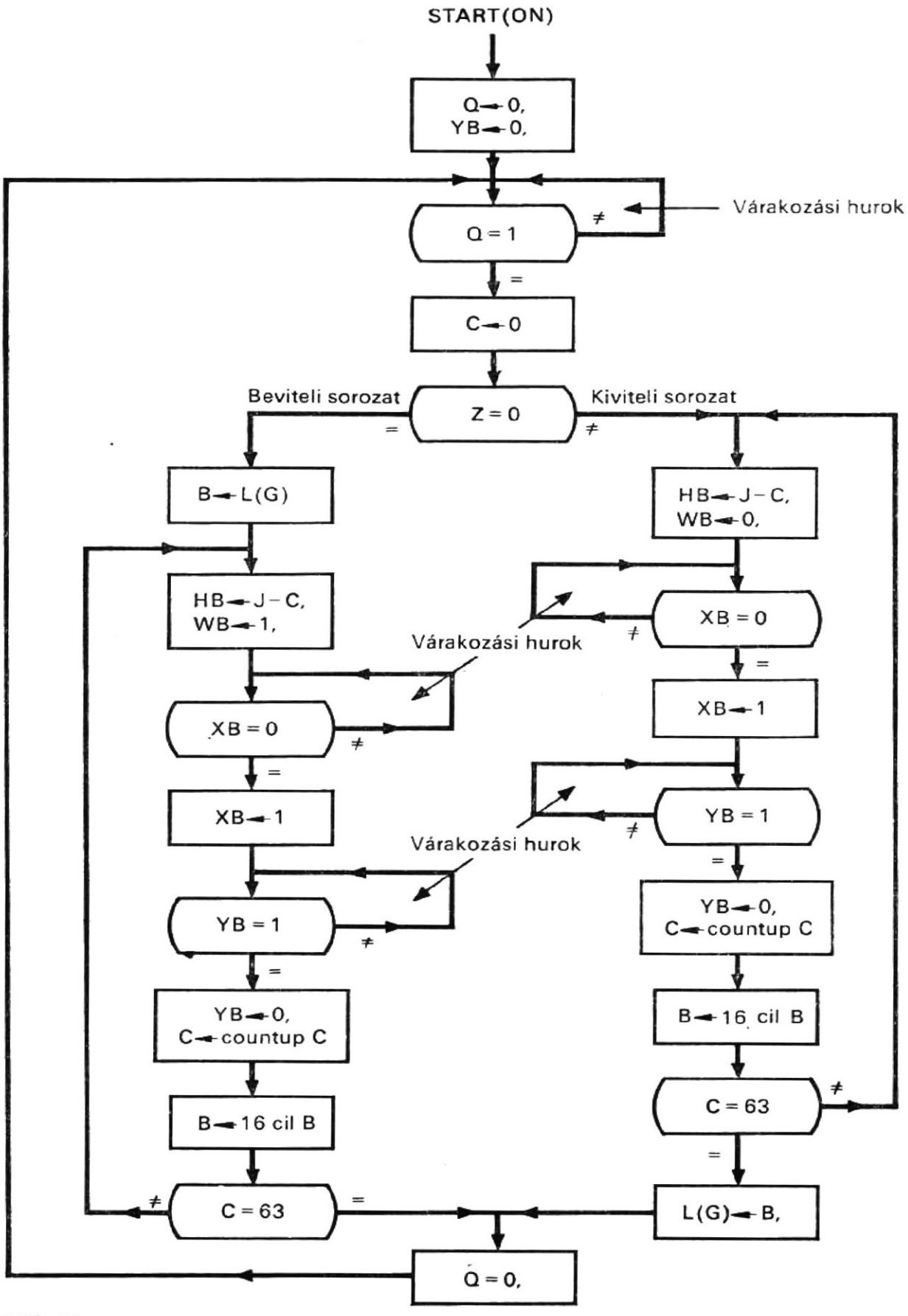
— egy 12-bites cím az L háttértár számára;
— egy 6-bites lapcím az M főtár számára.

Ez a két cím egy 16-bites szót képez, amelyet az M főtár tárol, és amelyre a beviteli vagy kiviteli utasítás címmezőjében levő cím mutat. Röviden valamely bevitel- vagy kivitelátvitel egy utasítást és egy hozzárendelt címszót igényel.

Az IOU a következő funkcionális egységekből áll: a B tartalom- (puffer-) regiszter, a HB címregiszter, a J lapcímregiszter, a G címregiszter, az S vezérlőregiszter és a C számláló. Ehhez járul még 4 egybites regiszter: a WB, YB, Q és Z. Ha a WB regiszter 0, olvasási műveletet kezdeményez a főtárban; egyébként pedig írási műveletet ír elő. Ha az YB regiszter 1, ez jelzi, hogy a kívánt tárművelet befejeződött. A Q regiszter 1 értéke arra mutat, hogy az IOU foglalt; egyébként pedig szabad. Ha a Z regiszter 1, azt jelöli, hogy a CPU kiviteli műveletsorozatot kezdeményez; egyébként pedig beviteli sorozatra van szüksége. Egyszerűség kedvéért feltételezzük, hogy a háttértár ciklusideje megegyezik az órajelgenerátor ciklusidejével. (Azt az esetet, amikor ezek nem egyformák, a feladatokban szerepeltetjük.)

Az alábbiakban leírjuk az IOU regisztereit és funkcionális elemeit.

```
Comment, az IOU struktúrája                                        (8.51)
Register,     HB(0—11),        $címregiszter
              WB,              $olvasási művelet a főtárban, ha 1;
                               egyébként írási műveletet jelöl
              YB,              $tárművelet befejeződött, ha értéke 1
              S(0—3),          $IOU vezérlőregiszter
              Q,               $az IOU foglalt, ha értéke 1; egyébként IOU
                                 szabad
              Z,               $CPU kiviteli sorozatot kezdeményez, ha 1;
                                 egyébként beviteli sorozat szükséges
              B(0—1023),       $pufferregiszter
              J(0—5),          $lapcímregiszter
              C(0—5),          $számláló
              G(0—9),          $címregiszter
Subregister,  B(BUF)=B(0—15),
Decoder,      KS(0—15)=S,
Memory,       L(G)=L(0—1023, 0—1023),
Clock,        P,
```



8.21. ábra
A be-, ill. kiviteli egység folyamatábrája

Az IOU sorrendi működése a 8.21. ábrán látható. Az IOU szabad, ill. foglalt, ha a Q regiszter értéke 0, ill. 1. Amikor az IOU aktivvá válik mind a Q, mind az YB regisztert 0-ra állítják. Ezután megvizsgálják a Q regisztert, hogy az értéke 1-e. Ha nem, folyamatosan vizsgálják, amíg csak 1 nem lesz. Ezt az ismételt vizsgálatot a 8.21. ábrán a várakozási hurok jelöli. Amikor a Q 1 lesz, a C számlálót 0-ra állítják. Most már megkezdődhet egy beviteli vagy kiviteli műveletsorozat.

Amikor a Z regiszter értéke 0, beviteli művelet sorozatot kezdeményez. Ez azzal kezdődik, hogy egy szót kiolvasnak az L háttértárból a B pufferregiszterbe (a szó helyét a G címregiszter határozza meg). A J-G regisztersor tárolja a főtár címét azon a helyen, ahová a B (BUF) részregiszterben levő pillanatnyi 16 bitet át kell vinni. Ezt a címet a J-G regisztersorból viszik át a H Bcímregiszterbe. A WB regiszter 1 értéket kap az olvasási művelet vezérlésére és ezután a kérés megy az M főtárhoz oly módon, hogy az XB 1 értéket kap, ha a tartalma 0 volt. Az IOU várakozási hurokba lép, ha az XB értéke nem 0. Ha az XB 0, az IOU egy másik várakozási hurokba lép, mindaddig vizsgálva az YB regisztert, amíg az 1 nem lesz. Ha az YB értéke 1, azt jelzi, hogy a kívánt tárművelet befejeződött. Az IOU ezt úgy nyugtázza, hogy az YB regisztert visszaállítja 0-ra.

Ezután a C számláló tartalmát 1-gyel megnövelik. A B regisztert 16 bitpozícióval ciklikusan balra léptetik. Ezt követően megvizsgálják, hogy a C 0-e. Ha nem 0, megismétli a J-G regisztersorban levő főtárcímnek a HB regiszterbe való átvitelét, a WB és az XB regisztereknek 1 érték adását és így tovább, amíg a C el nem éri a 63 értéket. Ekkor a B regiszterben levő 1024-bites szó egyetlen lapként átvitelre került az M főtárba. Figyeljük meg, hogy az M főtárban a lap első helyét a J lapcímregiszter, míg a „lokális címeket" a C számláló szolgáltatja. Ezzel befejeződött az L háttértárból az M főtárba történő átvitel. A Q regisztert visszaállítják 0-ra, jelezve, hogy az IOU ismét szabad és az IOU visszatér mindaddig a várakozási hurokba, amíg a Q regiszter értéke újból 1 nem lesz.

Amikor a Z regiszter értéke 1, akkor kiviteli sorozat kezdeményezéséről van szó. Egy 64-szavas lapot olvasnak ki az M főtárból a B pufferregiszterbe és azután az L háttértárban tárolják. Amint a 8.21. ábrán látható, ezen adatátvitelhez szükséges mikroműveletek hasonlók a bemeneti sorozatban levőkhöz. Amikor ez az adatátvitel befejeződött, a Q regisztert ismét 0-ra állítják, annak jelzésére, hogy az IOU ismét szabad. Az IOU visszatér a várakozási hurokba mindaddig, amíg a Q regiszter értéke ismét 1-et nem tartalmaz.

A 8.21. folyamatábrán bemutatott IOU szervezést a következőkben CDL utasításokkal is leírjuk.

```
Comment, az IOU működése                                        (8.52)
/START(ON)/  Q←0, YB←0, S←8,
/KS(8)*P/    IF (Q←0) THEN (S←10),
/KS(10)*P/   C←0, IF (Z=0) THEN (S←6) ELSE (S←14),

Comment, bemeneti műveletsorozat

/KS(6)*P/    B←L(G), S←7,
/KS(7)*P/    HB←J-C, WB←1, S←2,
/KS(2)*P/    IF (XB=0) THEN (XB←1, S←3),
/KS(3)*P/    IF (YB=1) THEN (S←1),
/KS(1)*P/    YB←0, C←countup C, S←5,
/KS(5)*P/    B←16 cil B, IF (C≠63) THEN (S←7) ELSE (S←12),

Comment, kimeneti műveletsorozat

/KS(14)*P/   HB←JC, WB←0, S←4,
/KS(4)*P/    IF (XB=0) THEN (XB←1, S←15),
```

```
/KS(15)*P/   IF (YB=1) THEN (S←11),
/KS(11)*P/   YB←0, C←countup C, S←9,
/KS(9)*P/    B←16 cil B, IF (C≠63) THEN (S←14) ELSE (S←13),
/KS(13)*P/   L(G)←B, S←12,
/KS(12)*P)/  Q←0, S←8,
             END
```

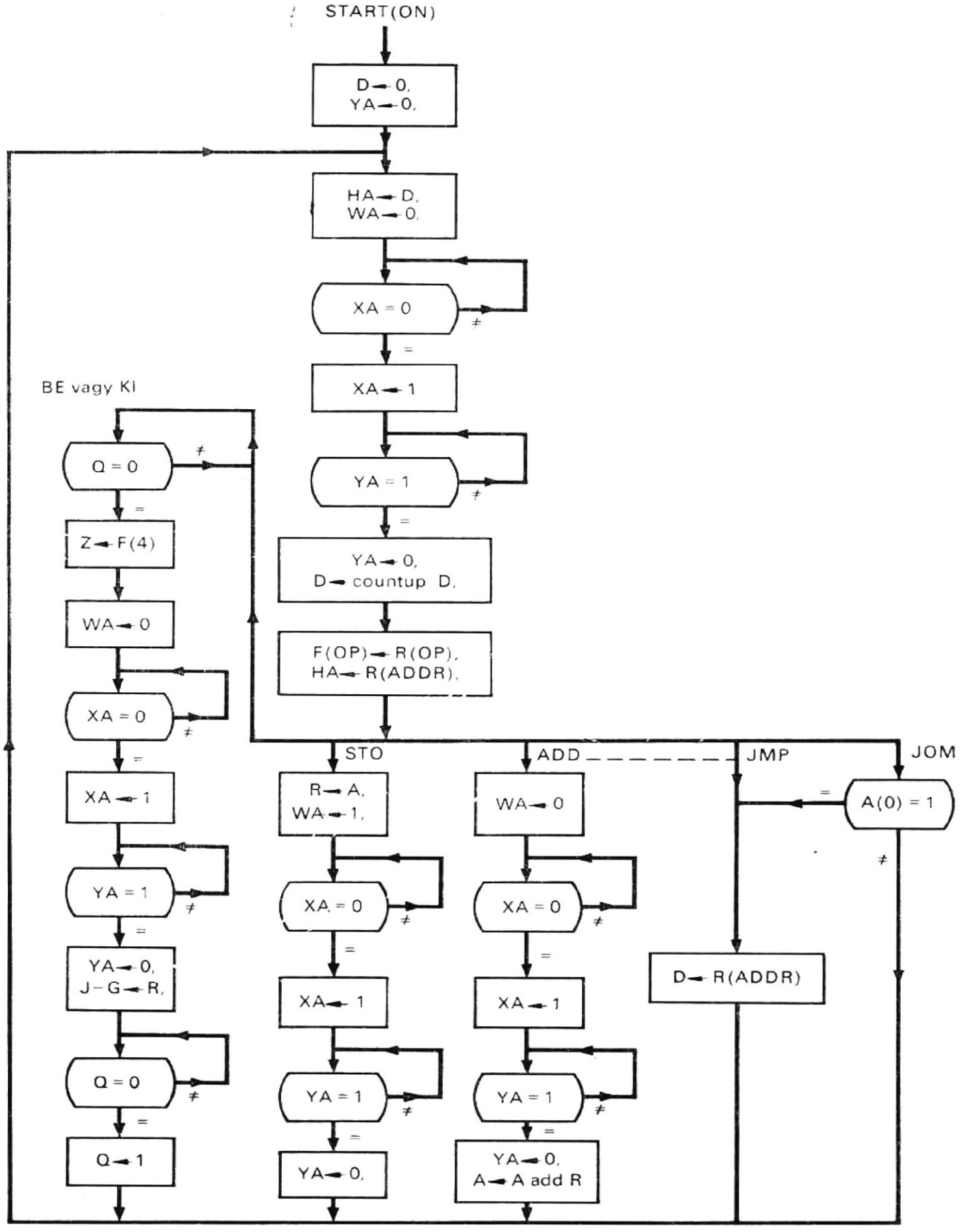
8.4.3. Központi feldolgozóegység

A központi feldolgozóegység az A akkumulátorból, az R tartalcm- (puffer-) regiszterből, a HA címregiszterből, a D programregiszterből, a C számlálóból és az F fázisvezérlő regiszterből áll. Ezekhez járul még két egybites regiszter: a WA és az YA. A WA regiszter 0-ás állapotban olvasási műveletet vezérel; 1-es állapotban pedig írási műveletet. Az YA regiszter 1 értéke azt mutatja, hogy a kívánt tárművelet befejeződött. Ezek a regiszterek a 8.19. ábrán láthatók, míg a többi CPU-hoz tartozó funkcionális elemet a következő deklarációs utasításokkal írjuk le.

```
Comment, a CPU struktúrája                                        (8.53)
Register,     A(0—15),          $akkumulátor
              C(0—15),          $számláló
              R(0—15),          $adatregiszter
              HA(0—15),         $címregiszter
              D(0—11),          $utasításregiszter
              F(0—4),           $CPU vezérlőregiszter
              WA,               $olvasási művelet, ha értéke 0; egyébként
                                  írási művelet
              YA,               $ha 1, befejeződött a CPU-tól érkező tár-
                                  művelet
Subregister,  F(OP)=F(1—4),
              R(OP)=R(0—3),
              R(ADDR)=R(4—15),
Decoder,      KF(0—31)=F,
Switch,       START(ON),
Clock,        P,
```

A CPU sorrendi működése a 8.22. folyamatábrán látható. Miután a START kapcsolóval működésbe hozták, a CPU utasításlehívó műveletsorozattal kezdi működését és ezután ágazik el valamelyik műveletsorozathoz, amelyet az utasításlehívó sorozat alatt dekódolt műveleti kód határoz meg. Valahányszor a CPU a tárhoz fordul az utasításlehívó vagy az utasításvégrehajtó sorozat alatt, belép egy várakozási hurokba. Ez idő alatt az YA regisztert folyamatosan vizsgálják. Ha az YA regiszter értéke azt jelzi, hogy a kívánt tárművelet befejeződött; ekkor a CPU újra kezdi működését.

Amint a 8.22. ábrán látható, amikor a START kapcsoló ON állapotba kerül, a D és az YA regiszterek 0 értéket kapnak. Az utasításlehívó ciklus a D regiszter tartalmának a HA címregiszterbe való átvitelével kezdődik. Ez azután 0 értéket ad a WA regiszternek, annak jelzésére, hogy ez olvasási művelet, és utasításhoz tárhozzáférés szükséges oly módon, hogy az MU-ban levő XA regiszter 1 értéket kap, ha az XA-ban 0 volt. Ha az XA nem volt 0, a CPU mindaddig várakozik, amíg értéke 0 nem lesz. A CPU most belép a tár várakozási hurokba. Ha az YA regiszter értéke 1, a CPU ezt az YA regiszternek 0-ra való visszaállításával nyugtázza és azután a D regisztert



8.22. ábra
A központi feldolgozóegység folyamatábrája

1-gyel dekrementálja. Most a R(OP) és R(ADDR) regiszterek tartalmát — amely a műveleti kód, ill. az operandus címe — átviszik az F(OP) részregiszterbe, ill. a HA regiszterbe. Ekkor valamely adott utasításvégrehajtó sorozatot specifikáló műveleti kód az F fázisregiszterben van, míg az operanduscím a HA címregiszterben. Ezzel az utasításlehívó ciklus befejeződött. Figyeljük meg, hogy az első utasítást az M tár első helyéről hívták le, mivel a D regiszter értéke kezdetben 0 volt.

Ha a műveleti kód oktális 15 vagy 16, az F(4) bit bemeneti műveletsorozatot jelöl, ha értéke 0, és kimeneti sorozatot, ha értéke 1. A CPU belép a várakozási hurokba, folyamatosan vizsgálva a Q regisztert. Amikor a Q regiszter 0-át tartalmaz, ez azt jelzi, hogy az IOU szabad; egyébként pedig foglalt. Amikor az IOU szabad, az F(4) bit tartalmát átviszik az IOU-ban levő Z regiszterbe. A CPU az MU-hoz fordul, az MU-ban levő XA regiszternek 1 értéket ad, ha az XA tartalma 0 és olvasási műveletet kezdeményez, és a WA regiszternek 0 értéket ad. Az IOU most belép a tárvárakozási hurokba. Amikor az YA regiszter értéke 1 lesz, a CPU ezt az YA-nak 0-ra történő visszaállításával nyugtázza és az R regiszter tartalmát átviszi az IOU-ban levő J-G regisztersorba. Az IOU-t most működésbe hozták, a Q regiszternek 1 értéket adva, ha a Q tartalma előzőleg 0 volt. Ha a Q nem 0, a CPU mindaddig várakozik, amíg a Q ismét 0 nem lesz. A bemeneti vagy kimeneti műveletsorozat indul el és a CPU az utasításlehívó sorozattal folytatja működését. A műveletsorozatok a többi utasítás esetén is hasonlók.

A fenti CPU műveletsorozat részlegesen látható a 8.22. ábrán. Most ezt is leírjuk CDL utasításokkal is.

```
Comment, a CPU működésének leírása                                    (8.54)
/START(ON)/          D←0, YA←0, F←17,

Comment, utasításlehívó sorozat

/KF(17)*P/           HA←D, WA←0, F←8,
/KF(8)*P/            IF (XA=0) THEN (XA←1, F←16),
/KF(16)*P/           IF (YA=1) THEN (F←0),
/KF(0)*P/            D←countup D, YA←0, F←4,
/KF(4)*P/            F(OP←R(OP), HA←R(ADDR),

Comment, be-, ill. kiviteli műveletsorozat

/(KF(13)+KF(14))*P/  Z←F(4), F←12,
/KF(12)*P/           IF (Q=0) THEN (F←30),
/KF(30)*P/           WA←0, F←28,
/KF(28)*P/           IF (XA=0) THEN (XA←1, F←31),
/KF(31)*P/           IF (YA=1) THEN (F←23),
/KF(23)*P/           J-G←R, YA←0, F←28,
/KF(28)*P/           IF (Q=0) THEN (Q←1, F←17),

Comment, összeadáshoz tartozó műveletsorozat

/KF(6)*P/            WA←0, F←15,
/KF(15)*P/           IF (XA=0) THEN (XA←1, F←22),
/KF(22)*P/           IF (YA=1) THEN (F←20),
/KF(20)*P/           A←A add R, YA←0, F←17,

Comment, kivonást végző műveletsorozat

/KF(5)*P/            WA←0, F←21,
/KF(21)*P/           IF (XA=0) THEN (XA←1, F←7),
/KF(7)*P/            IF (YA=1) THEN (F←19),
/KF(19)*P/           A←A sub R, YA←0, F←17,
```

```
Comment, load (betöltés) műveletsorozat
/KF(2)*P/          WA←0, F2←4,
/KF(24)*P/         IF (XA=0) THEN (XA←1, F←18),
/KF(18)*P/         IF (YA=1) THEN (F←25),
/KF(25)*P/         A←R, YA←0, F←17,

Comment, store (tárolás) műveletsorozat
/KF(1)*P/          R←A, WA←1, F←14,
/KF(14)*P)         IF (XA=0) THEN (XA←1, F←3),
/KF(3)*P/          IF (YA=1) THEN (F←27),
/KF(27)*P/         YA←0, F←17,

Comment, ugrás mínuszra (jump-on-minus) műveletsorozat
/KF(9)*P/          IF (A(0)=1) THEN (F←10), ELSE (F←17),

Comment, ugrás (jump) műveletsorozat
/KF(10)*P/         D←R(ADDR), F←17,
                   END
```

8.4.4. Illesztések (interface)

A (8.54) utasítás három független rendszertechnikai egységet ír le, amelyek közül mindegyiknek saját vezérlése van. Tehát három vezérlési ciklus jön létre, amelyek függetlenül és aszinkron módon működnek. Az MU állandóan kiszolgálja a CPU-t és az IOU-t. Ha a CPU egyszer működésbe hozza az IOU-t, az magára marad, egy beviteli vagy kiviteli műveletsorozat végrehajtására. Ez idő alatt a CPU a főtárban tárolt programot hajtja végre.

A három rendszertechnikai egység az illesztésen keresztül tart kapcsolatot egymással. Három illesztés van: egy a központi feldolgozóegység és a be-, ill. kiviteli egység között, egy másik a tárolóegység és a központi feldolgozóegység között, és a harmadik a tárolóegység és a be-, ill. kiviteli egység között. Ezek az illesztések a fentiekben leírt mikroműveletek által igényelt címhez, adathoz, állapothoz és vezérléshez szükséges vezetékek és jelek segítségével jönnek létre. Ezen interface-eket a 8.23. ábra mutatja. Ha az órajelgenerátor megosztott, további órajel- vagy időzítővezetékek szükségesek. 19 vezeték van a CPU és az IOU között. Kettő közülük vezérlést végez az I/O (B/K)-sorozatot kezdeményező CPU mikroműveletekben és azt jelzi, hogy a kívánt sorozat bevitel-e vagy kivitel:

Q←1 és Z←F(4).

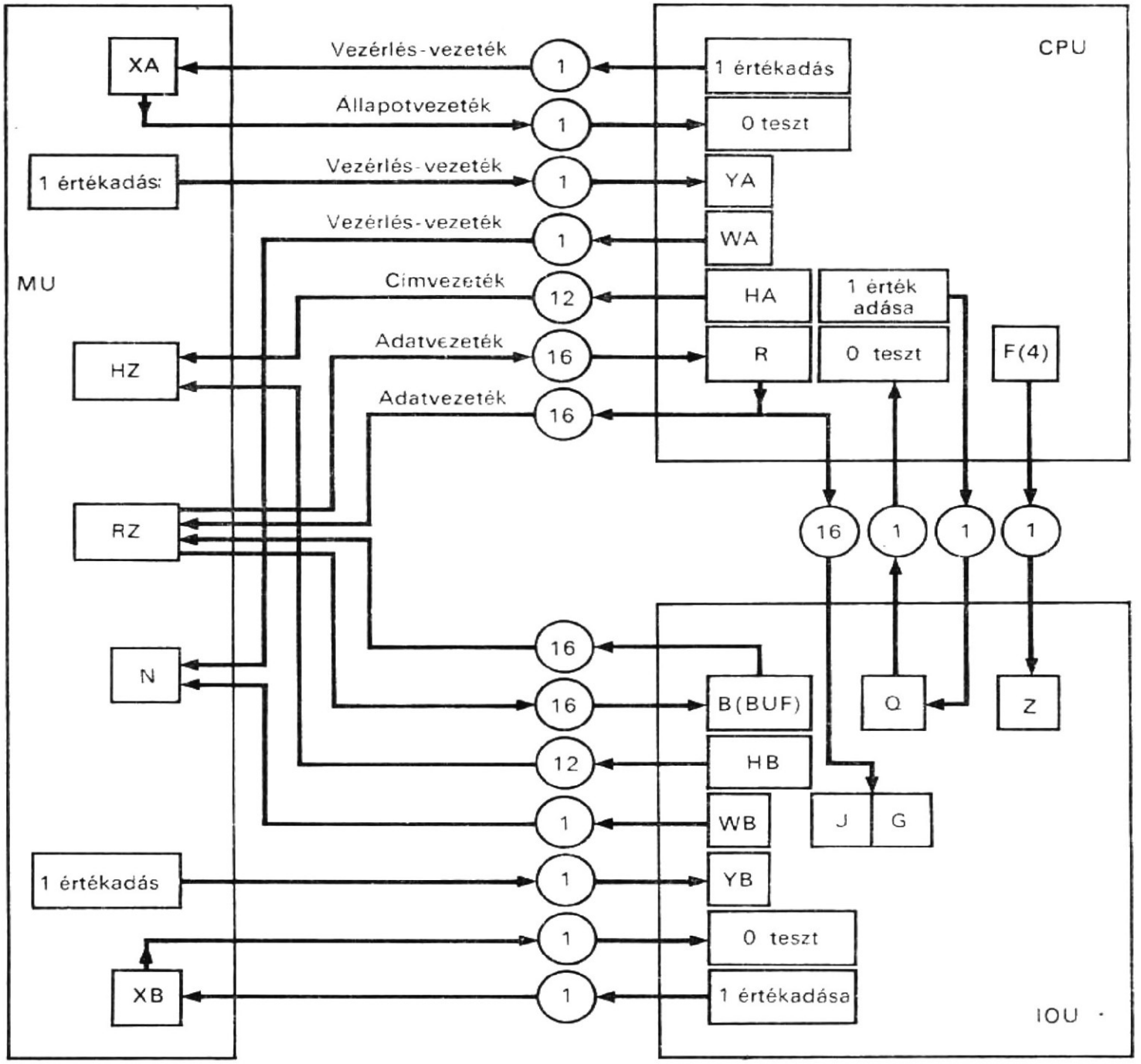
Egy állapot- (státusz-) vezeték a CPU-nál megvizsgálja:

Q=0.

A maradék 16 vezeték címvezeték, a következő, a CPU-ból az IOU-ba történő címet átvivő CPU mikroművelethez:

J—G←R.

Összesen 48 vezeték van a CPU és az MU között: 12 címvezeték, 32 adatvezeték, 1 állapotvezeték és 3 vezérlést végző vezeték. A 3, vezérlést végző vezeték a következő mikroműveletekből adódik: tárolóállapot-kérés a CPU-tól, annak jelzése, hogy az



8.23. ábra
Interface-ek a tárolt programú számítógép rendszertechnikai egységei között

MU befejezte-e a hozzáférést, valamint annak jelzése, hogy a kívánt tárművelet olvasás-e vagy írás:

XA←1, YA←1 és N←WA.

Az állapot- (státusz-) vezeték a következő vizsgálati feltételt valósítja meg:

XA=0.

A 12 címvezeték a következő mikroutasításhoz tartozik (amely a tárcímet a CPU-ból az MU-ba viszi át):

HZ←HA.

Az adatvezetékek a CPU-ban levő R regiszter és az MU-ban levő RZ regiszter közötti adatátviteli mikroműveletekhez tartoznak:

R←RZ és RZ←R.

Hasonlóképpen összesen 48 vezeték van az MU és az IOU között.
A fenti három illesztést most a következő terminálutasításokkal írhatjuk le.

```
Comment, a CPU és az IOU között interface                    (8.55)
Terminal,   R(0—5),      $lapcím-vezetékek az IOU J regiszteréhez
            R(6—15),     $szócímvezetékek az IOU G regiszteréhez
            F(4),        $a CPU-ból jövő bevitel vagy kivitel jelzése
            Q,           $az IOU állapotvezetéke
            IO-KÉR,      $a CPU I/O kérés vezetéke

Comment, az MU és a CPU közötti illesztés                    (8.56)
Terminal,   XA,          $tár (MU) állapotvezetéke
            MEM-KÉR-A,   $a CPU tárolókérés vezetéke
            KÉSZ-A,      $művelet kész jelzés az MU-ból
            WA,          $a CPU-ból érkező olvasás vagy írás kérés
            HA(0—11),    $a CPU-ból érkező tár-címvezetékek
            RZ(0—15),    $adatátviteli vezeték az MU-ból
            R(0—15),     $adatbeviteli vezeték a CPU-ból

Comment, az MU és az IOU közötti illesztés                   (8.57)
Terminal,   XB,          $tár állapotvezetéke
            MEM-KÉR-B,   $a CPU-tól érkező tárkérés vezetéke
            KÉSZ-B,      $művelet kész vezetékek az MU-ból
            WB,          $az IOU-ból érkező olvasás vagy írás kérés
            HB(0—11),    $az IOU-ból érkező tárcímvezetékek
            RZ(0—15),    $adatkiviteli vezetékek (az MU-ból)
            B(BUF),      $adatbeviteli vezetékek (az IOU-ból)
```

8.5. A számítógéprendszer konfigurációja

Az eddig leírt számítógéprendszernek három rendszertechnikai egysége van, egy valóságos számítógéprendszernek azonban általában több. Legtöbbször olyan további rendszertechnikai egységek is vannak, mint pl. a csatorna, vezérlőegység és be- ill. kiviteli berendezés. Ebben a szakaszban bemutatjuk ezt a három rendszertechnikai egységet és szerepüket a számítógéprendszer működésében.

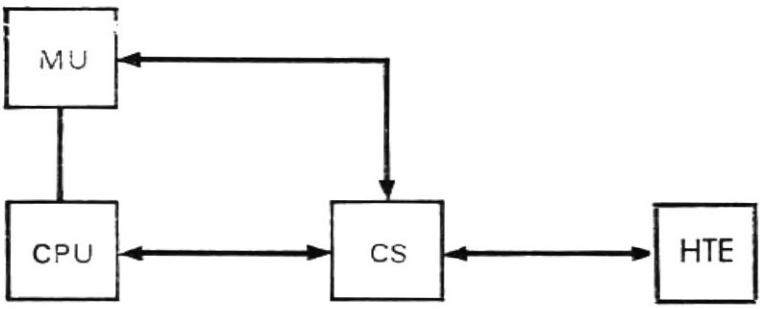
8.5.1. Csatorna

Az előző fejezetben olyan számítógéprendszert írtunk le, amelynek három rendszertechnikai egysége volt: az MU, a CPU és az IOU. Az IOU egyebek között az L háttértárat is tartalmazza. Ha két vagy több háttértárra van szükség, ezek a tárak megoszthatják egymás között a parancskiadás, az átmeneti tárolás, átvitel, számlálás és vezér-

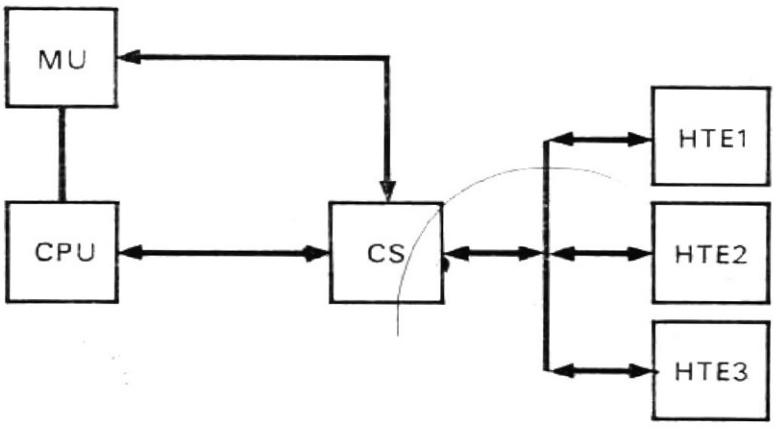
lés műveleteit végző hardware-t. E hardware-megosztás megvalósítására az IOU-t két rendszertechnikai egységbe osztják, amint az a 8.24. ábrán látható. Az egyik rendszertechnikai egység lényegében az L háttértárat tartalmazza és *háttértároló egységnek* vagy HTE-nek nevezik, míg a másik rendszertechnikai egység a további elemeket tartalmazza és *csatornának* (CS-nek) nevezik. Ha a három háttértárat egy számítógéprendszerhez kell illeszteni, és ha egyidőben csak egy érhető el közülük, akkor ezt a három HTE-t a számítógéprendszerhez a 8.25. ábrán látható módon, a csatornán keresztül csatolják. Ebben a fejezetben a 8.24. ábrán bemutatott csatornaszervezését írjuk le.

A csatorna alapvetően az MU főtárbeli címet tároló HB regiszterből, a HTE címet tároló G regiszterből, az MU lapcímet tároló J regiszterből, a B adatregiszterből, a C számlálóból, és a WB, YB, Q és Z egybites regiszterekből áll. Ezek a regiszterek ugyanazok, mint amilyeneket a (8.51) utasításban írtunk le, azzal az eltéréssel, hogy a B regiszternek most csak 16 bitje van. A fenti funkcionális elemeket a 8.26. ábrán láthatjuk és az alábbiakban írjuk le.

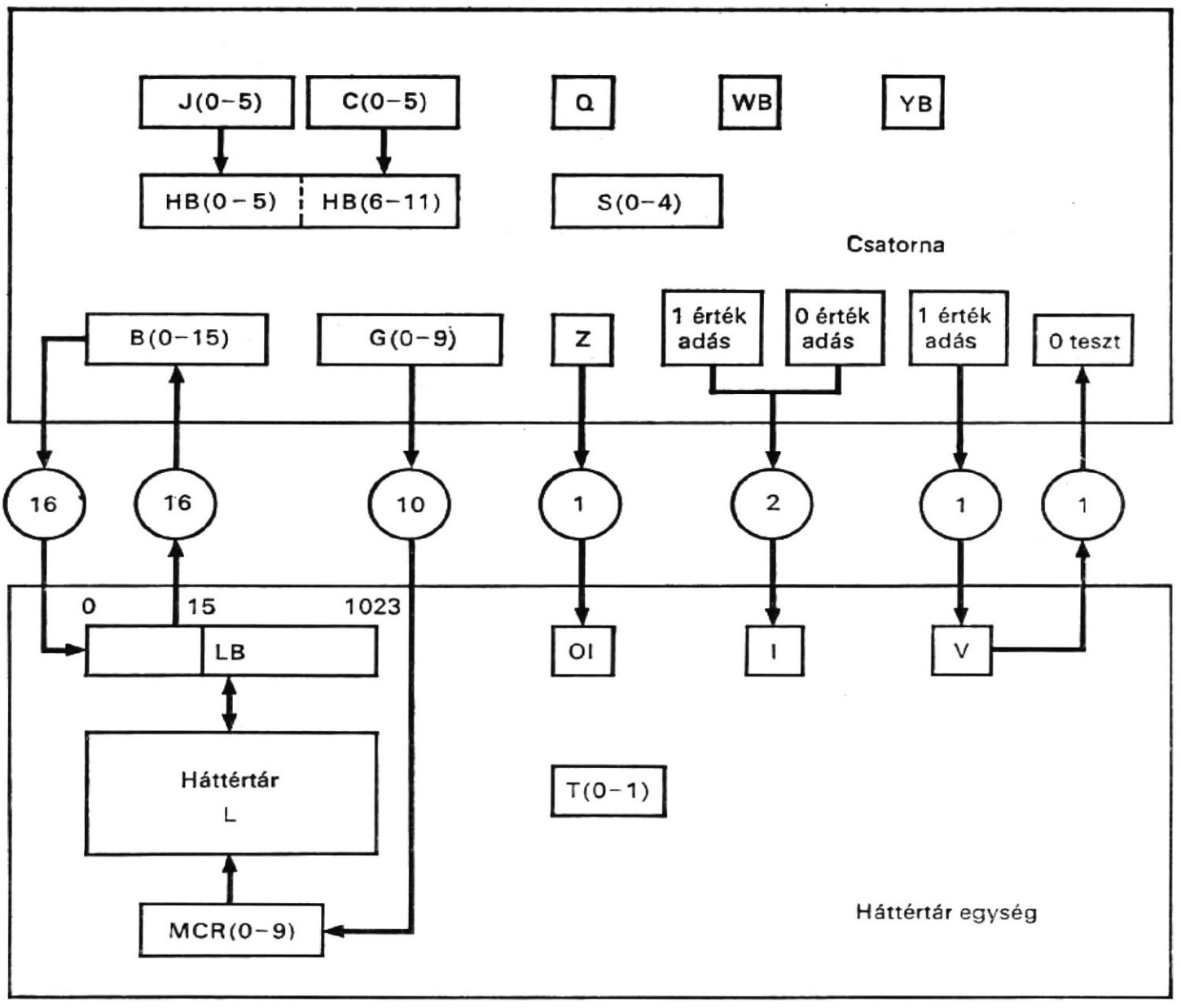
```
Comment, a csatorna struktúrája                                    (8.58)
Register,     HB(0—11),      $főtárbeli cím regisztere
              B(0—15),       $adatregiszter
              G(0—9),        $a HTE címregiszter
              J(0—5),        $lapcímregiszter
              C(0—5),        $számláló
              S(0—3),        $csatornavezérlő regiszter
              WB,            $ha 0, olvasás; egyébként írás,
              YB,            $ha 1, befejeződött a tárművelet
              Q,             $ha 1, a CS foglalt, egyébként a csatorna szabad
              Z,             $ha 1, a műveletsorozat kivitel, egyébként
                               bevitel
Subregister,  B(BUF)=B(0—15),
Decoder,      KS(0—15)=S,
Clock,        P,
Switch,       START(ON),
```



8.24. ábra
CS csatornával és HTE háttértároló egységgel rendelkező számítógéprendszer



8.25. ábra
Egy CS csatornával és három, HTE1, HTE2 és HTE3 háttértároló egységgel rendelkező számítógép rendszer



8.26. ábra
A csatorna és a háttértároló egység struktúrája

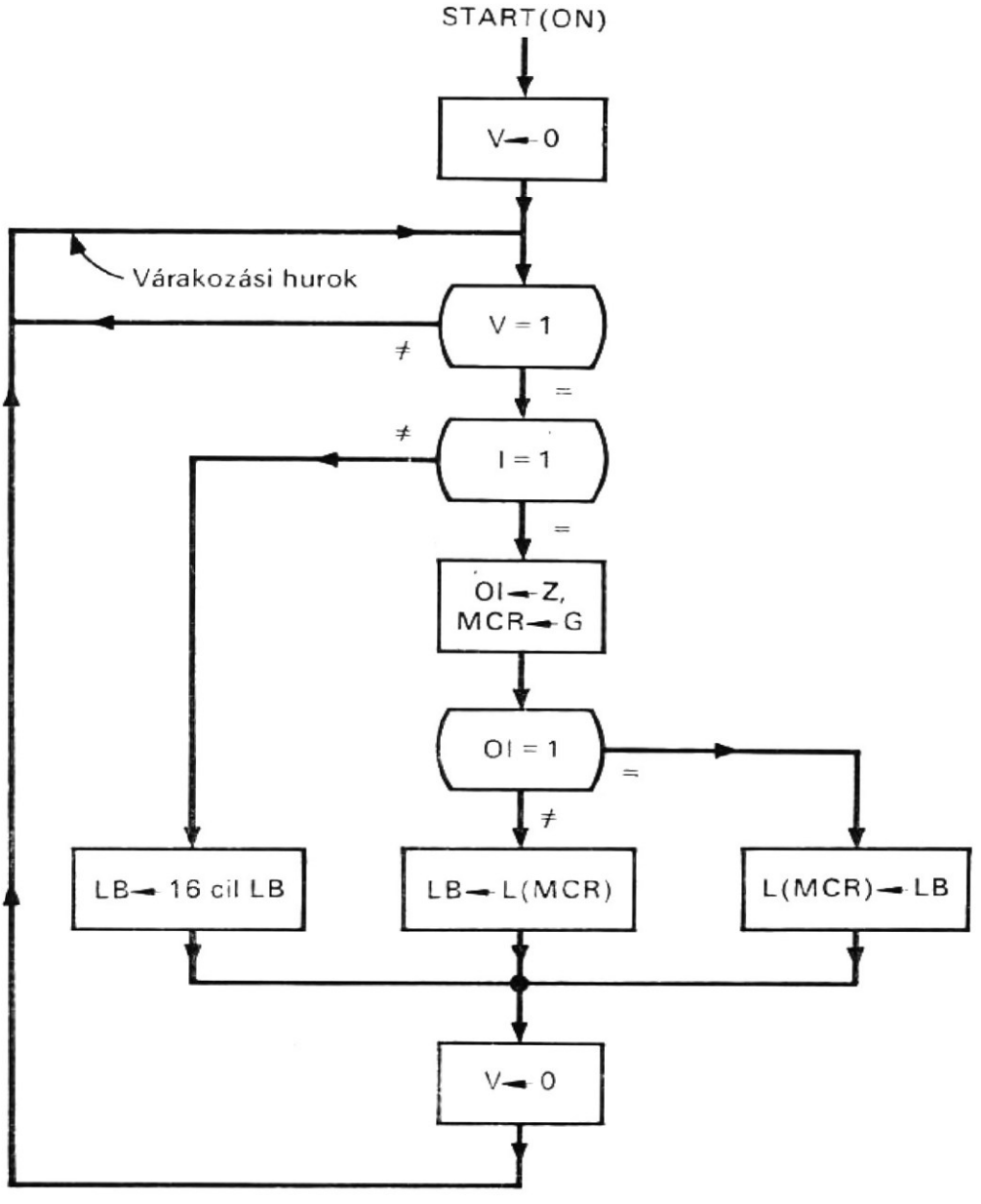
A háttértároló egység az L háttértárból, az MCR címregiszterből és az LB átmeneti tár- (puffer-) regiszterből áll. Továbbá tartalmazza a T regisztert és három egybites regisztert, ezek jele: az OI, I és V. A T regiszter az órajelgenerátorral együtt a HTE működési sorrendjét határozza meg. Az OI regiszter azt jelzi, hogy művelet olvasás-e (azaz bevitel), vagy pedig írás (azaz kivitel). Az I regiszter határozza meg, hogy a művelet 16-bites átvitel-e, vagy pedig olvasás—írás művelet. A V regiszter a HTE működését kezdeményezi, a HTE-t tehát a V regiszter vezérli. Mindezek a funkciók a 8.26. ábrán láthatók és az alábbiakban írjuk le őket.

```
Comment, a háttértár struktúrája                                    (8.59)
Register,   LB(0—1023),    $HTE átmeneti tár- (puffer-) regiszter
            MCR(0—9),      $HTE címregiszter
            T(0—1),        $HTE vezérlőregiszter
            OI,            $ha 0, olvasás; egyébként írás
            V,             $ha HTE műveletsorozat kezdeményezés
            I,             $ha 1, adatátvitel van, egyébként HTE írás—
                             olvasás művelet
```

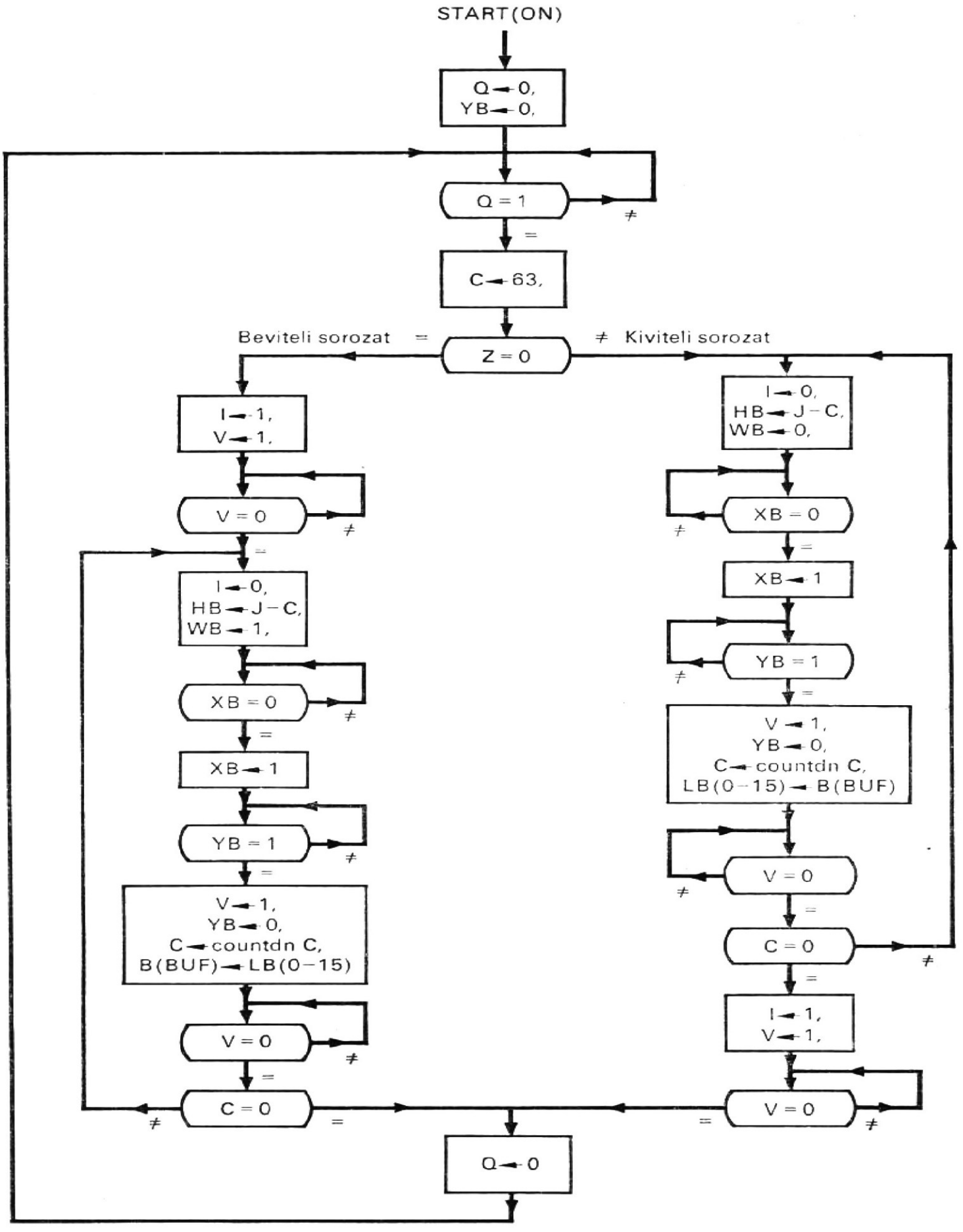
Memory, L(MCR)=L(0—1023, 0—1023),
Decoder, KT(0—3)=T,
Clock, P,
Switch, START(ON),

A HTE működését a 8.27. folyamatábrán láthatjuk. Amint látható, mindannyiszor HTE műveletsorozat indul, amikor a V regiszter 1 értéket kap. Ha az I regiszter 0, ez adatátviteli műveletet jelez, ekkor az LB regiszter tartalmát ciklikusan balra léptetik 16 bitpozícióval. Ha az I regiszter 1, ez olvasási vagy írási műveletet jelez, attól függően, hogy az OI regiszter 0, ill. 1 értéket tartalmaz-e. Ha az OI regiszter 1, az LB regiszterben levő adatszót beviszik az L tárba; egyébként pedig egy adatszót olvasnak ki az L tárból az LB regiszterbe. Mindhárom említett esetben a sorozat a V regiszternek 0-ra állításával fejeződik be.

A csatorna működését a 8.28. folyamatábra mutatja be. Ez hasonlít az IOU működéséhez, amelyet a 8.21. folyamatábrán láthattunk. A leglényegesebb különbség az, hogy most az L háttértár elérésére szolgáló, valamint a ciklikusan balra léptető mikroműveleteket a HTE-ben hajtják végre és a CS vezérli ezeket a V és I regiszterek segítségével. Továbbá adatátvitel szükséges a HTE és a CS között; ezeket a CS kezdeményezi.



8.27. ábra
A háttértároló egység működésének folyamatábrája



8.28. ábra
A csatorna működésének folyamatábrája

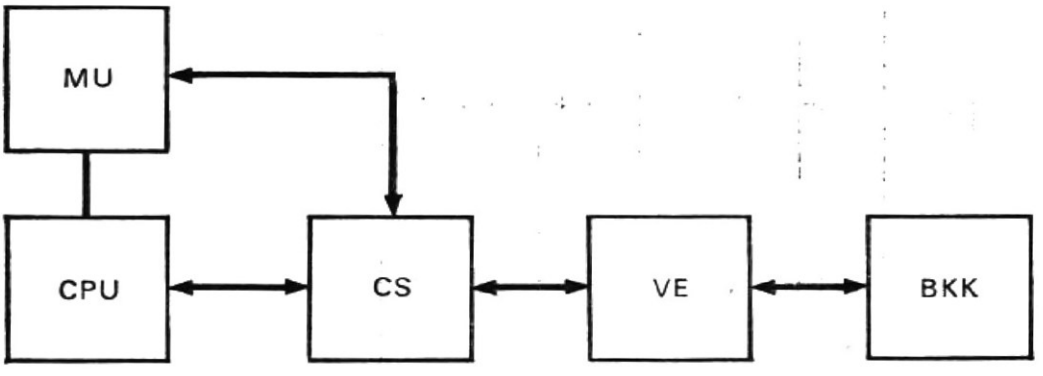
Amint a 8.26. ábrán látható, a CS és a HTE között 47 vezeték van. Ezeket a vezetékeket összefoglalóan *B/K illesztéseknek* (I/O interface) nevezik, mivel a B/K (I/O) készülékek kapcsolódnak ezekhez a vezetékekhez. A 8.24. ábrán bemutatott számítógéprendszerben az MU és a CS közötti illesztés ugyanaz marad, mint a 8.23. ábrán látható MU és IOU közötti. A CPU és a CS közötti illesztés is ugyanaz marad, mint a CPU és az IOU közötti illesztés.

A 8.25. ábra számítógéprendszerében három HTE kapcsolódik a csatornához. Ez azt jelenti, hogy most egy további mező szükséges a B/K (I/O) utasításban a HTE specifikálására (azaz kell egy be-, ill. kimeneti készülék cím). A csatorna kialakítható úgy, hogy az olvasás és írás műveletektől eltérő parancsokat hajtson végre. A parancskódot, a be-, ill. kimeneti készülék címét, a főtár azon címét, ahol az adatokat elérik, valamint a további szükséges információkat egyetlen csatornaparancsszóban lehet összefoglalni, amelyet a csatornának kell végrehajtania. Az is megvalósítható, hogy a csatorna egy csatornaprogramot hajtson végre, amely nem más, mint a csatornaparancsszavak sorozata. A csatornaprogramot a főtárban lehet tárolni, vagy egy olyan tárban, amely a csatorna része. Tehát a csatornafunkciók növekedhetnek és a csatorna olyan feldolgozóegységgé válhat, amely egyébként a CPU által végrehajtandó B/K (I/O) funkciókat a CPU-val egyidejűleg végzi el; ez a szimultán működés jelentősen javítja a CPU kihasználtságát.

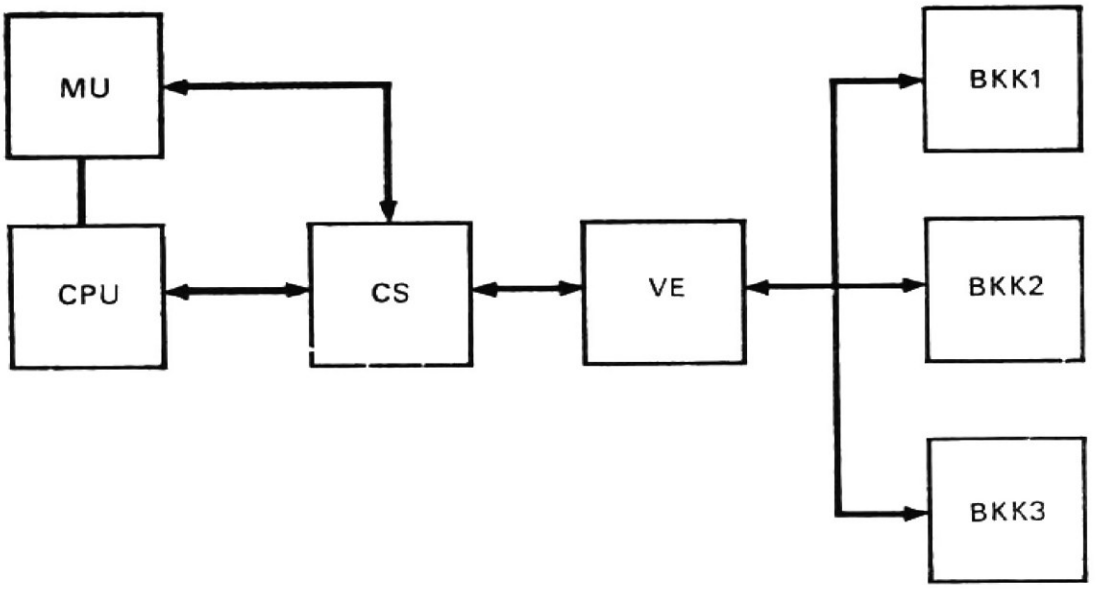
8.5.2. Vezérlőegység

Az olyan be-, ill. kiviteli egység, mint pl. a 8.24. ábrán látható HTE, rendszerint két egységből épül fel, a VE vezérlőegységből (**C**ontrol **U**nit: CU) és a BKK be-, ill. kimeneti készülékből (**I**nput **O**utput **D**evice: IOD), mivel a BKK általában elektromechanikus, míg a VE elektronikus. A 8.29. ábrán különálló VE-t és BKK-t tartalmazó számítógéprendszer látható. A VE-re és BKK-ra való szétválasztás lehetővé teszi, hogy egyetlen vezérlőegységgel egy vagy több egyforma vagy hasonló BKK-t lehessen vezérelni. A 8.30. ábrán egy VE-vel és három BKK-val rendelkező számítógéprendszer látható. Ez a rendszer azonban egyidőben csak egy BKK működését teszi lehetővé.

Ha kiterjesztük azt az elképzelést, hogy több BKK-t lehet egyetlen vezérlőegységhez csatolni, akkor egyetlen csatornához is több vezérlőegységet csatlakoztathatunk. Ilyen felépítésű számítógéprendszer látható a 8.31. ábrán. Az, hogy a BKK-khoz külön vezérlőegységeket kell használni, gyakran amiatt van, hogy a különféle BKK típusokhoz különböző típusú vezérlőegységekre van szükség. Azonban a 8.31. ábrán látható



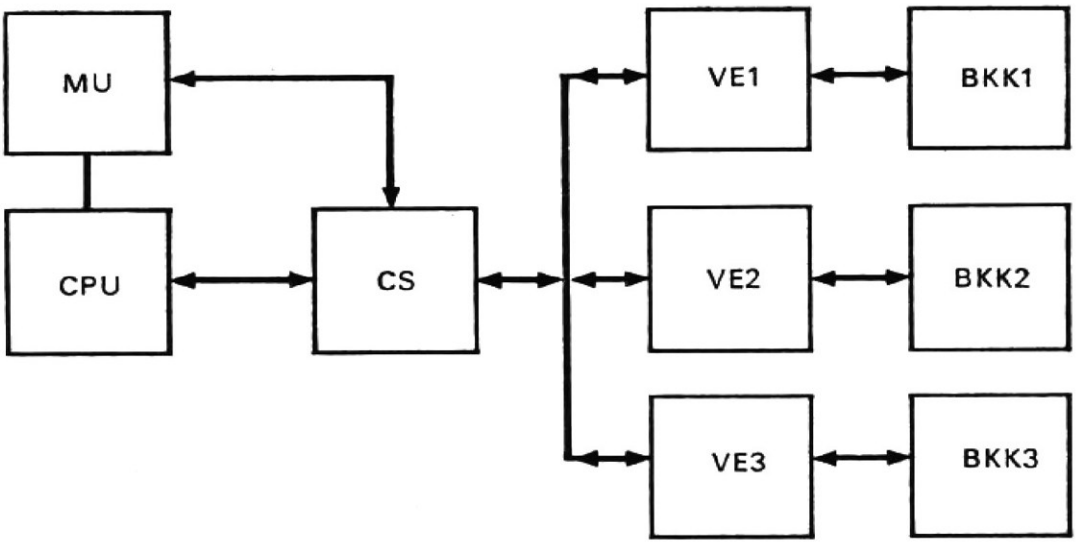
8.29. ábra
Csatornával, vezérlőegységgel és egy be-, ill. kiviteli készülékkel rendelkező számítógéprendszer



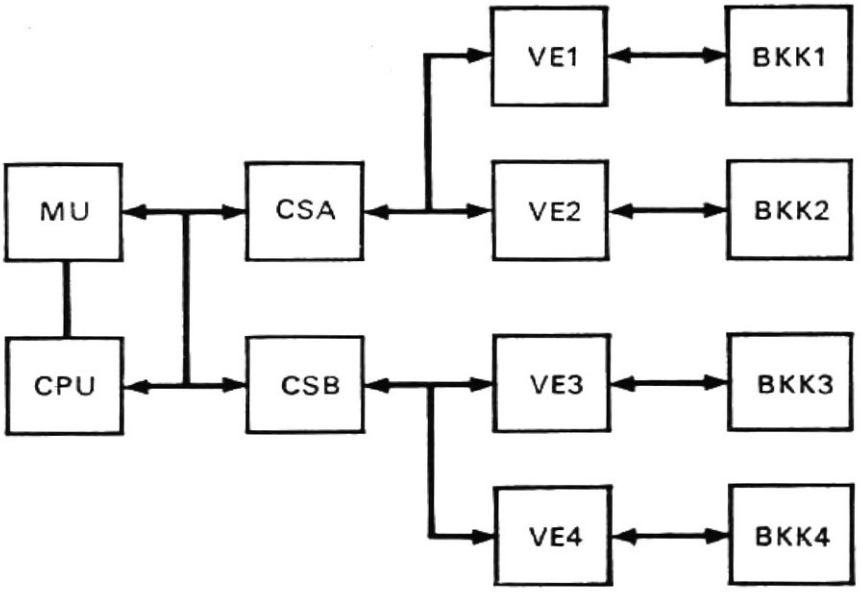
8.30. ábra
Csatornával, vezérlőegységgel és három be-, ill. kiviteli készülékkel rendelkező számítógéprendszer

elrendezés csak egy vezérlőegységnek és egy be-, ill. kimeneti készüléknek teszi lehetővé, hogy a rendszer tárolóegységével kommunikáljon. Ha azt akarjuk, hogy egynél több BKK tudjon a tárolóegységgel kommunikálni, egynél több csatorna szükséges. A 8.32. ábrán kétcsatornás számítógéprendszer látható, ahol a VE1 és VE2 vezérlőegységek az A csatornához, a VE3 és a VE4 vezérlőegységek pedig a B csatornához csatlakoznak. Ebben a rendszerben két BKK tud egyszerre kommunikálni az MU-val.

A fentiekben leírt csatorna egyszerre egy BKK-t kezel. Vannak olyan csatornák, amelyek időosztásos üzemmódban működtethetők több, kissebességű BKK kiszolgálására. Az ilyen időosztásos működésmódot *multiplexelésnek* nevezik, az ilyen csatornát pedig *multiplexcsatornának*, amint azt a későbbiekben, a 9. és 10. fejezetekben eírjuk.



8.31. ábra
Számítógéprendszer három vezérlőegységgel, amelyek közül mindegyikhez egy be-, ill. kiviteli készülék tartozik

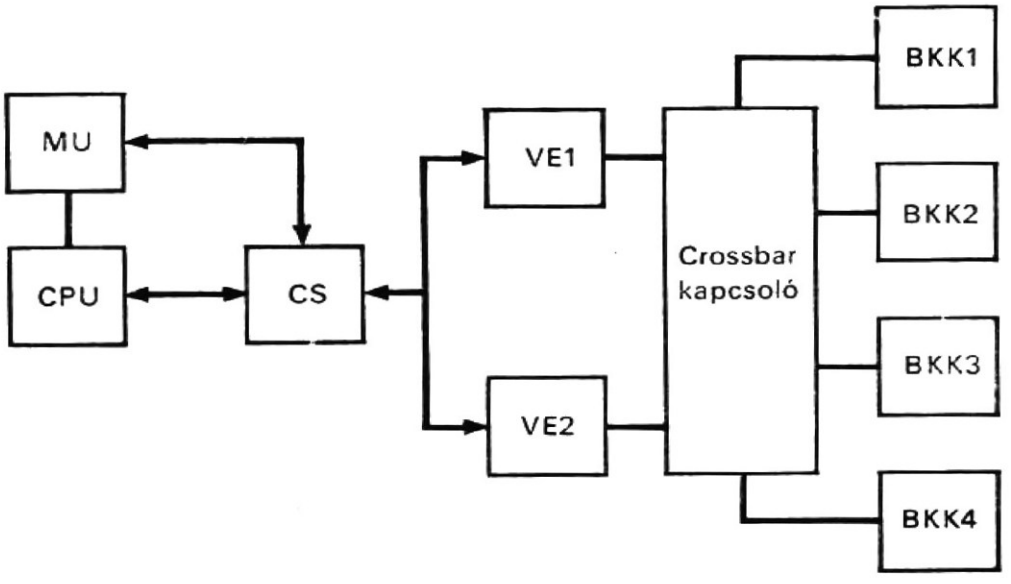


8.32. ábra
A és B csatornával rendelkező számítógéprendszer. Mindegyik csatornának két vezérlőegysége van, mindkettőhöz egy be-, ill. kiviteli készülék tartozik

8.5.3. Moduláris konfiguráció

A 8.32. ábrán látható számítógéprendszer csatornáinak, vezérlőegységeinek és be-, ill. kiviteli készülékeinek összekapcsolása fához hasonló struktúrát mutat. Mindegyik BKK-t a központi feldolgozóegységgel vagy a tárolóegységgel el lehet érni, de ennek a hozzáférésnek meghatározott útvonala van és keresztül kell haladnunk egy adott csatornán és vezérlőegységen ahhoz, hogy elérjük a be-, ill. kiviteli készüléket. Ha a CS vagy a VE foglalt (vagy hibás), akkor ehhez a vezérlőegységhez vagy ehhez a csatornához kapcsolt egyéb BKK mindaddig nem lesz elérhető, amíg az útvonal szabaddá nem válik (vagy meg nem javítják).

A be-, ill. kiviteli készülékek elérhetősége javítható a CPU, ill. az MU közötti crossbar kapcsoló alkalmazásával. A 8.33. ábrán olyan számítógéprendszer látható, amelyben a vezérlőegységeket és a be-, ill. kimeneti készülékeket 2×4-es crossbar kapcsoló köti össze, ahol a két VE közül bármelyik el tudja érni a négy BKK mindegykét.



8.33. ábra
Crossbar kapcsolót tartalmazó számítógéprendszer. Mindegyik vezérlőegységgel (VE) elérhető a négy be-, ill. kiviteli készülék (BKK) bármelyike

A 8.34. ábrán olyan számítógéprendszer látható, amelynek egy 2×4-es és egy 2×2-es crossbar kapcsolója van. Ebben a rendszerben a 2×4-es crossbar kapcsoló a vezérlőegységeket és a be-, ill. kimeneti készülékeket, míg a 2×2-es crossbar kapcsoló a csatornákat és a vezérlőegységeket kapcsolja össze. A négy BKK közül bármelyik elérhető bármelyik csatorna által, és egyidőben két BKK tud működni.

Egy másik, két crossbar kapcsolóval rendelkező számítógéprendszer látható a 8.35. ábrán. A 4×4-es crossbar kapcsoló a két központi feldolgozóegységet és a két csatornát kapcsolja a négy tárolóegységhez; ez a kapcsoló akár a CPU, akár a CS részére lehetővé teszi a négy MU közül bármelyikhez való hozzáférést. A 2×*n*-es crossbar kapcsoló a csatornák és a vezérlőegységek között teremt kapcsolatot; e kapcsoló segítségével bármely csatorna el tudja érni az *n* darab BKK bármelyikét. Bármely tárolóegység (MU), központi feldolgozóegység, csatorna, vezérlőegység, vagy be-, ill. kimeneti készülék meghibásodása nem teszi tönkre a teljes számítógéprendszer működését, csak esetleg csökkenti a teljesítőképességét. A számítógéprendszer rendszeregységeinek modularitása miatt nevezhető a 8.35. ábrán látható rendszer *moduláris felépítésűnek*.

Feladatok

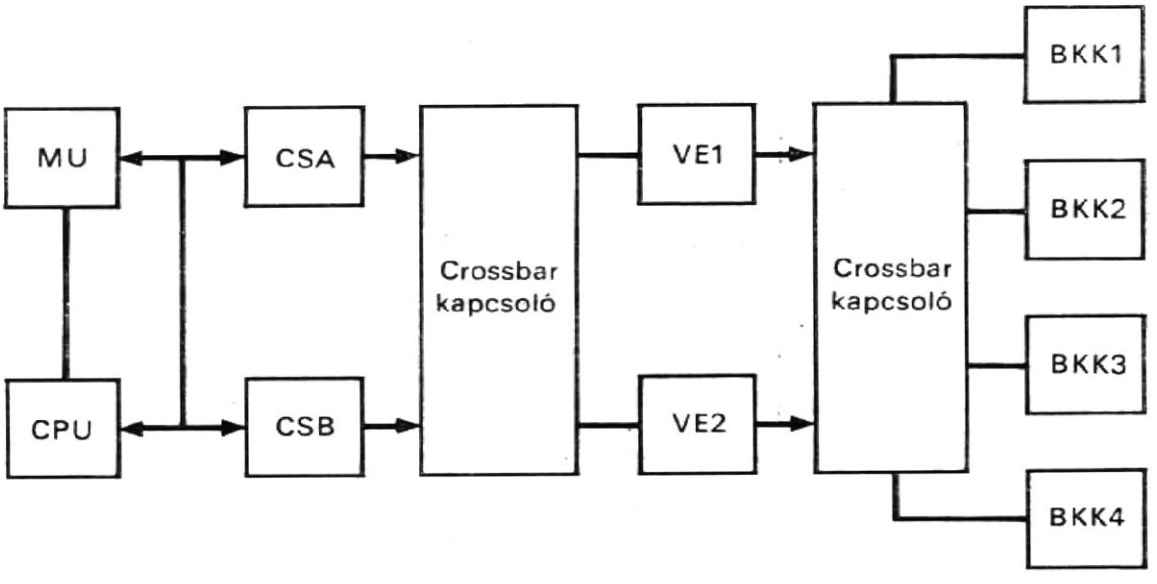
8.1. Rajzoljuk fel a (8.1), (8.3) és (8.5) utasításokban leírt vezérlések hardware-megvalósítását és elemezzük a különbségeiket!

8.2. Rajzoljuk fel a (8.2), (8.4) és (8.6) utasításokban leírt vezérlések hardware-megvalósítását és elemezzük a különbségeiket!

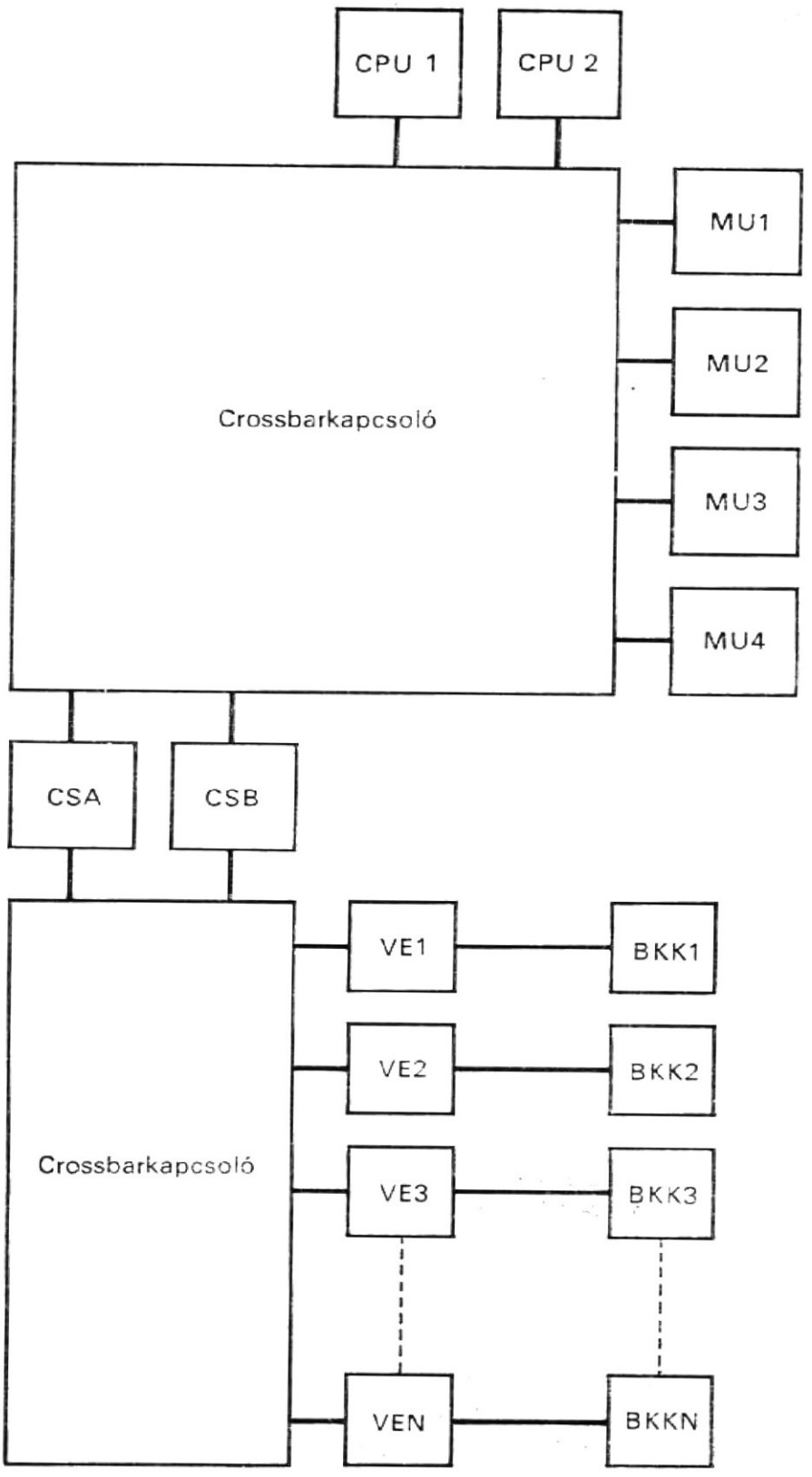
8.3. A 8.36. ábrán megadunk egy állapotdiagramot, amelyen minden bekarikázott szám egy állapotot képvisel. Készítsünk olyan, huzalozott sorrendi logikával ellátott vezérlést, amely a diagram állapotai szerint előírt sorrendben működik!

8.4. A (8.18)...(8.21) utasításokban leírt mikroprogramok alapján készítsük el a három mikroutasítást, ezek szóhossza 32 bit!

8.5. Határozzuk meg a mikroprogramozott CPU algoritmusát és mikroprogramját, ha az összehasonlítandó bináris számok előjellel és abszolút értékkel vannak ábrázolva!



8.34. ábra
Két crossbar kapcsolóval rendelkező számítógéprendszer. Mindegyik csatornával a négy be-, ill. kiviteli készülék (BKK) bármelyike elérhető

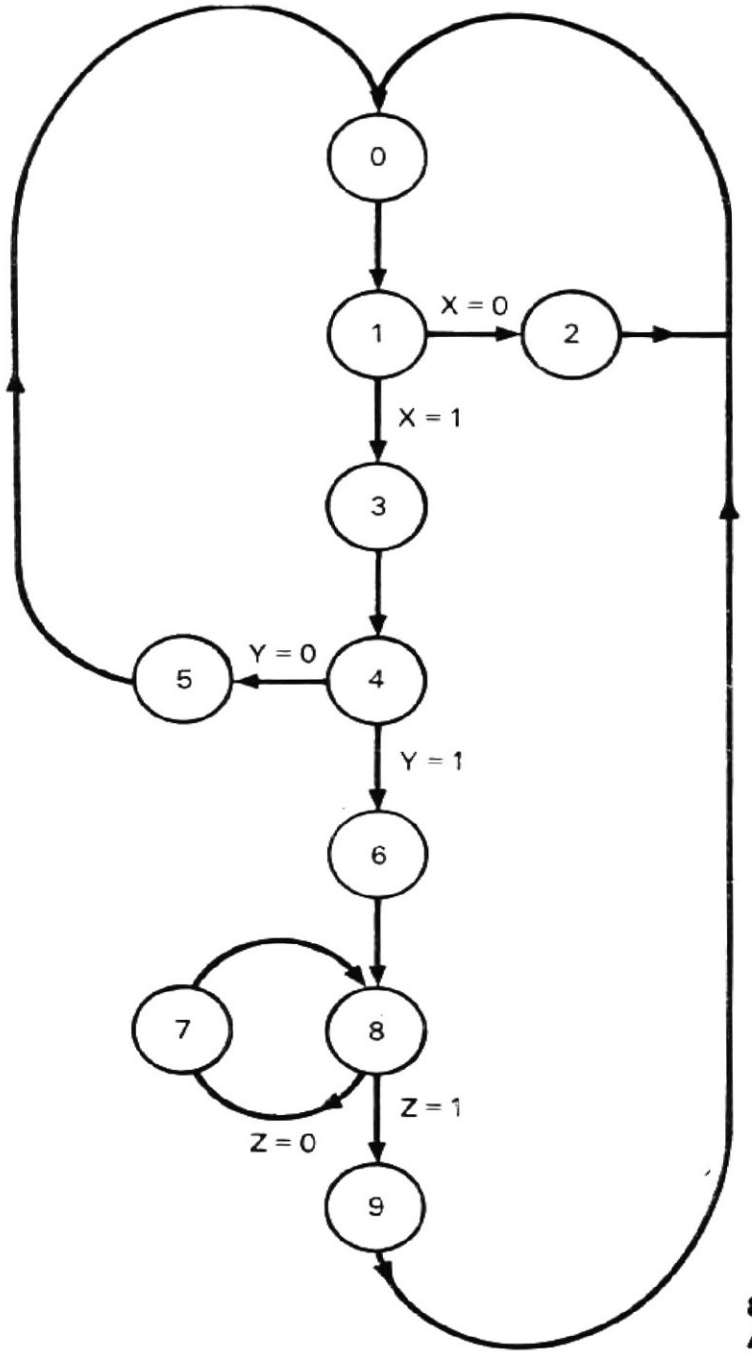


8.35. ábra
Két crossbar kapcsolós számítógéprendszer

8.6. Készítsünk mikroprogramot az A regiszterben levő bináris számnak a B regiszterben levő bináris számmal való összeszorzására, feltételezve, hogy ezek a bináris számok előjellel és abszolút értékkel vannak ábrázolva! Ha szükséges, alkalmazzunk további funkcionális elemeket és műveleteket!

8.7. Módosítsuk a (8.25) utasításokban leírt műveletsorozatot azon feltétel alapján történő ugrásra, amikor az akkumulátorban levő szám kisebb, mint az MQ regiszterben tárolt szám!

8.8. Módosítsuk a (8.27)...(8.30) utasításokban leírt címzést úgy, hogy olyan ismétlő utasítást is engedélyezünk, amely az utasítás végrehajtását egy számlálómezőben specifikált szorzószám szerint ismétli!



8.36. ábra
Állapotdiagram

8.9. Rajzoljunk állapotdiagramot a 8.20. folyamatábrából!
8.10. Ismételjük meg a 8.9. feladatot a 8.21. folyamatábrára!
8.11. Ismételjük meg a 8.9. feladatot a 8.22. folyamatábrára!
8.12. Egészítsük ki a 8.22. folyamatábrát az összes többi utasításra!
8.13. Tegyük fel, hogy ugyanaz a P órajelgenerátor áll a CPU, IOU és MU rendelkezésére. Ha az M főtár négy órajelperiódust igényel a tárciklus végrehajtásához és ha az L háttértárnak 10 órajelperiódusra van szüksége, hogyan módosulnak a (8.50), (8.52) és (8.54) utasítások?
8.14. Amikor akár a CPU, akár az IOU a 8.20....8.22. ábrán olvasást vagy írást kér a tártól, keletkezik-e sorbaállás? Ha igen, hogyan lehetne hosszabb sort kezelni?
8.15. Írjuk le a CDL-ben a háttértárolóegység (HTE) sorrendi működését, hasonlóan ahhoz, hogy azt a (8.59) utasítás és a 8.27. folyamatábra leírja! A leírás tartalmazza az időzítő- és vezérlőjeleket!
8.16. Ismételjük meg a 8.15. feladatot a (8.58) utasítások és a 8.28. folyamatábra felhasználásával!

8.17. A 8.24. ábrán látható számítógéprendszer háttértárolóegysége (HTE) és csatornája (CS) között 16-bites szélességű adatút viszi át az adatokat. Abban az esetben, ha az adatút 4-bites szélességű:
a) módosítsuk a HTE és CS struktúráját! Használjuk fel a 8.26. ábra blokkdiagramját és a (8.58) és (8.59) utasításokat!
b) Módosítsuk a 8.27. és 8.28. ábrákon bemutatott műveletsorozatokat!
c) Módosítsuk a HTE és CS közötti illesztést, felhasználva a 8.26. ábrát!

8.18. Írjuk le CDL-ben a 8.25. ábrán látható számítógéprendszer csatornájának és HTE-inek struktúráit és műveletsorozatait! Tételezzük fel, hogy a B/K utasításban van egy mező, amely az adott HTE-t specifikálja. A tár és a csatorna közötti adatátviteli út szélessége 16 bit és a HTE és a csatorna közötti adatátviteli út szélesség 4 bit.

Irodalom[11]

[1] MERSEL, J.: *Program Interrupt on the Univac Scientific Computer.* Proc. of the WJCC, 1956, pp. 52... ...53.

[2] BROOKS, F. P., JR.: *A Program-Controlled Program Interruption System.* Proc. of the Easter Computer Conference, 1957, pp. 128...132.

[3] TURNER, L. R.—RAWLINGS, J. H.: *Realization of Randomly Timed Computer Input and Output by Means of an Interrupt Feature.* IRE Trans. on Elec. Comp., June, 1958, pp. 141...149.

[4] BECKMAN, F. S.—BROOKS, F. P., JR—LAWLESS, W. J., JR.: *Developments in the Logical Organization of Computer Arithmetic and Control Units.* Prog. of the IRE, January, 1961, pp. 53...66.

[5] BUCHHOLZ, W., ed.: *Planning a Computer System.* New York, McGraw Hill Book Company, 1962.

[6] GRASSELLI, A.: *The Design of Program-modifiable Microprogrammed Control Units.* IRE Trans. on Elec. Comp., June 1962, pp. 336...339.

[7] BLAAUW, G. A.: *Multisystem Organization.* part V of „The Structure of System/360," IBM Systems Journal 3, Nos. 2 and 3, 1964, pp. 181/195....

[8] McGee, W. C.—Petersen, H. E.: *Microprogram Control for the Experimental Sciences.* Proc. of the FJCC, 1965, New York: Spartan Books, pp. 77...91.

[9] CAMPBELL, C. R.—NEILSON, D. A.: *Microprogramming the Spectra 70/35.* Datamation, September, 1966, pp. 64...67.

[10] GOUNTAIS, R. J.—VISS, N. J.: *A Method of Processor Selection for Interrupt Handling in a Multiprocessors System.* Proc. of the IEEE, December 1966, pp. 1812...1819.

[11] ELORES, I.: *Computer Organization.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1969.

[12] VANDLING, G. C.—WALDECKER, D. E.: *The Microprogram Control Technique for Digital Logic Design.* Computer Design, August, 1969. pp. 44...51.

[13] RAMAMOORTHY, C. V.—TSUCHIYA, M.: *A Study of User-microprogrammable Computers.* Proc. of the SJCC, 1970, pp. 165...181.

[11] A mikroprogramozásra vonatkozó hivatkozások a 3. fejezetben egyúttal ennek a fejezetnek is hivatkozásai.

Számítógép-szervezés

A számítógép-szervezés a digitális számítógéprendszer adat- és utasításformátumait-kódjait, funkcionális egységeit, struktúráját és algoritmusait írja le. Ez a fejezet bemutatja egy modern digitális számítógép rendszertechnikai egységeit, majd egy ismert számítógépről ad leírást. Ennek a számítógépnek a csatornáit a 10. fejezetben tárgyaljuk.

9.1. Rendszertechnikai egységek

A korszerű digitális számítógéprendszer sok egységből tevődik össze: a főtárból, a központi feldolgozóegységből, csatornákból, B/K illesztést megvalósító egységekből, B/K vezérlőegységekből és B/K készülékekből, mindezekhez egy működtető — operációs — rendszer is tartozik. A digitális számítógéprendszer rendszeregységeit mutatja be a 9.1. ábra.

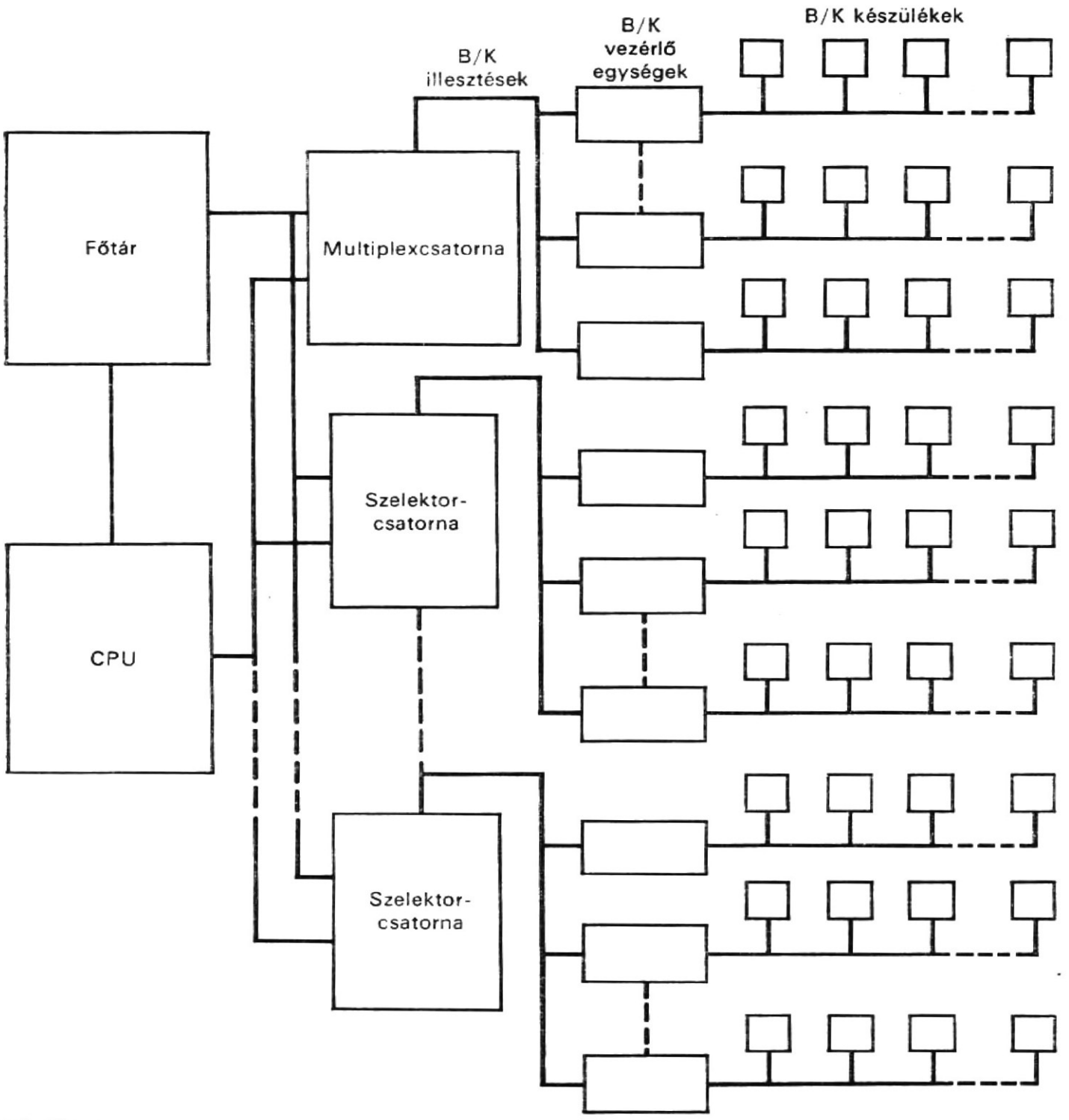
9.1.1. Főtár

A tárolt programú számítógép a főtárban tárolt program utasításainak és az adatainak sorrendi végrehajtását végzi. Két programtípus van:

— a rendszerprogram és
— a felhasználói program.

A rendszerprogramok a működtetőrendszert, a fordítóprogramokat, a szolgáltató (utility) programokat stb. foglalják magukban. Ezeket általában *rendszersoftware*-nek nevezik.

A főtár közvetlen elérésű (random access) tár. Ez képezi a tárolt programú számítógép és a számítógépek más típusai közötti legfontosabb különbséget. A korszerű, általános rendeltetésű digitális számítógép alkalmazásakor sok probléma merül fel,



9.1. ábra
Számítógéprendszer egységei

mivel a főtár kapacitása mindig kevésnek bizonyul a rendszer- és a felhasználói programok tárolására. A digitális számítógépek főtárának kapacitása az egészen korai digitális számítógépek néhány ezer szavától a modern, nagyszámítógépek néhány százezer vagy annál több szaváig terjedhet. A tártechnológia óriási fejlődése ellenére egyes esetekben a főtár kapacitása még mindig kicsinek bizonyul, nagymérvű bővítése pedig túlságosan drága. Ez az oka, hogy a főtárhoz mindig mágnesszalag-, mágnesdob- vagy mágneslemez- (disk) tárak járulnak, amelyek ugyan lassúbbak, de a kapacitásuk nagyobb. E különböző típusok alkalmazása (ezek a tárak nagy különbségeket mutatnak adatátviteli sebességükben, adatformátumukban és hozzáférési módszereikben) jellemzi a korszerű digitális számítógépek hardware-jében és software-jében megmutatkozó bonyolultságot.

9.1.2.
Központi feldolgozóegység

A központi feldolgozóegység, a KFE (**C**entral **P**roces **U**nit: CPU):

— a feldolgozó,
— a processzor- (processzáló-) egységből — ezt gyakran aritmetikai és logikai egységnek is nevezik és
— a fővezérlőegységből áll.

A fővezérlőegység végzi egy adott utasításban a mikroműveletek sorrendi vezérlését, egy programban az utasítások sorrend szerinti végrehajtását, valamint egyes számítógépekben egyik programról a másikra történő vezérlésátadást. Ugyancsak ez teszi lehetővé a legtöbb számítógépben a műveleti sorrend belső vagy külső jellel történő megszakítását. Az utasítás végrehajtásakor a fővezérlőegység hívja le az utasítást a főtárból, értelmezi azt és azután végrehajt egy mikroműveletekből álló sorozatot.

Valamely utasításban előírt műveleteket a feldolgozóegység végzi el. Ezek bináris vagy decimális aritmetikai műveletek, logikai műveletek, valamint összehasonlítási, konverziós, szerkesztő és bitellenőrző műveletek lehetnek. E műveletekkel feldolgozott adat fix hosszúságú, változó hosszúságú, vagy egy gépen belül mindkettő is lehet. Egy nagy digitális számítógépnek hatékony és széles körű utasításkészlete kell, hogy legyen, a változó funkciók minimális tárterületet igénylő végrehajtásához.

9.1.3.
Csatornák

A csatornák olyan egységek, amelyek segítségével egyidőben mehetnek végbe az adatfeldolgozó műveletek a CPU-ban és B/K adatok átvitele a be-, ill. kimeneti készülékekkel. Egy vagy több B/K vezérlőegység csatlakozhat az egyes csatornákhoz. A csatornákat és a B/K vezérlőegységeket összekötő kábelt *B/K illesztésnek* (I/O interface) nevezik, amint a 9.1. ábrán látható. Minthogy egy vagy több be-, ill. kimeneti készülék kapcsolódik mindegyik B/K vezérlőegységhez, egyetlen csatornához több készülék csatlakoztatható.

A csatornáknak két típusa van: a szelektorcsatorna és a multiplexcsatorna. A *szelektorcsatornával* a be-, ill. kimenő adatokat nagysebességű készülékről, pl. mágnesdobtárról lehet átvinni. Azért nevezik ezt a típust így, mert egyidőben csak egy B/K készüléket lehet kiválasztani — szelektálni — a csatornán. Amikor a csatornát kiválasztották, az átviteli műveletet nem lehet mindaddig megszakítani, amíg az be nem fejeződik. A *multiplexcsatorna* segítségével kissebességű be-, ill. kimeneti készülékekről (mint pl. kártyaolvasó vagy sornyomtató) lehet be-, ill. kimenő adatokat szimultán — időmultiplex üzemben — átvinni. Ezt a csatorna és a B/K illesztés multiplexelésével (azaz időosztásával) lehet elérni.

9.1.4.
Be-, ill. kimeneti készülékek és vezérlőegységek

A külső táreszközöket, valamint azokat a készülékeket, amelyeket a számítógép a külvilággal való kapcsolattartásra használ, be-, ill. kimeneti (B/K) készülékeknek nevezik. A B/K készüléknek működéséhez valamilyen vezérlőberendezésre van szüksége. Ezt a vezérlőberendezést *B/K vezérlőegységnek* nevezik. A B/K készülék vezérlőegységét rendszerint külön álló egységként vagy pedig a B/K készülék integrált részeként alakítják ki. Egyes vezérlőegységek csak egy készüléket tudnak vezérelni, míg mások több, hasonló B/K készülék vezérlésére is alkalmasak.

9.1.5.
Működtetőrendszer

A korszerű univerzális számítógéprendszer csak a működtetőrendszerrel tekinthető komplettnek. A működtetőrendszer programok összessége, amely a felhasználói programok előkészítését és végrehajtását végzi. Ezeket arra tervezték és implementálták, hogy javítsák mind a felhasználó, mind a számítógéprendszer produktivitását.

A működtetőrendszer programjait (vagy rutinjait) két kategóriába lehet osztani:

— feldolgozó- (processzáló-) programokra:
— vezérlőprogramokra.

A **feldolgozóprogramok** segítenek a felhasználónak a számítógéprendszer használatában és a programozásra fordított erők csökkentésében. A feldolgozási programokhoz tartoznak az assemblerek, fordítóprogramok, programgenerátorok, osztályozóprogramok és az olyan szolgáltató rutinok, mint pl. a lyukkártyáról lyukszalagra konvertáló programok. A **vezérlőprogramok** segítségével a számítógép úgy felügyel a rendszerre, vezérli és működteti a rendszert, hogy az a beadott munkák folyamatos áramlását — a gépkezelő minimális közbelépése nélkül — dolgozza fel. Vezérlőprogramok pl. a monitor, a betöltő, a B/K vezérlőrendszer, az ütemező, a megszakítási rutin és a diagnosztizáló program. A vezérlő- és a feldolgozóprogramok egy állandó (rezidens) rész kivételével, amely mindig a főtárban van, valamelyik tömegtárban tárolt rendszerkönyvtárban helyezkednek el. A nagyobb hatékonyság érdekében célszerű a működtetőrendszert a számítógéppel egyidejűleg megtervezni. Csak ily módon válhat a működtetőrendszer a számítógéprendszer integrált részévé. Napjainkban a számítógépek egyes rendszerorientált feladatait, amilyen pl. a tárvédelem, a privilegizált utasítások kezelése, a program áthelyezés (relocation), az automatikus megszakítás és az ütemidőzítő, hardware-rel oldják meg.

9.2.
Formátumok és kódok

Annak érdekében, hogy részletesebben leírhassuk a különböző számítógépek szervezését, az IBM System/360 számítógépcsaládját választottuk (a 25, 30, 40, 44, 50, 65, 67, 75, 85 és 91-es modellt) példának, egyrészt ezek a modellek sok alkalmazásban előfordulnak, széles körben használatosak, másrészt a hardware és software tervezésére és megvalósítására fordított, a számítógép-tervezésben eddig példa nélkül álló erőfeszítések miatt. Az IBM System/360-as család adat- és utasításformátumainak és kódjainak leírásával kezdjük, mert ezek a család összes modelljeiben azonosak.

9.2.1.
Adatformátumok

A hardware-rel fizikailag megvalósított, legkisebb információegység a *bit*. Az alapvető címezhető és feldolgozható információegység nyolc bitből áll, amelyet fogalmilag egy csoportban fognak össze; ez a *byte* (ehhez járul még egy páratlan paritásbit is, amelyet az egyes byte-okhoz rendelnek). A főtár kapacitását a byte-ok számával adják meg. A regiszterek, valamint a rendszeregységek közötti adatátviteli utak szélessége is byte-okban van megadva.

Az adatformátumokat a 9.2. ábra mutatja. Az információ lehet fix hosszúságú és változó hosszúságú. A fix hosszúságú információ egy byte, egy félszó, egy szó vagy egy kettősszó hosszú lehet. A félszó két, egymást követő byte-ból álló csoport. A szó négy, egymást követő byte csoportja. A kettősszó nyolc, egymást követő byte csoportja. Amint a 9.2. ábrán látható, a fixpontos szám vagy egy félszó, vagy egy szó (másképpen teljes szó), míg a lebegőpontos szám vagy egy szó, vagy egy kettősszó. A logikai adatok rögzített hosszúságúak és szavakban helyezkednek el. A főtárban tárolt, rögzített hosszúságú információnak helyes címhatárokon kell elhelyezkednie, amint azt a következőkben bemutatjuk. A byte-ok bármilyen címen elhelyezhetők, a félszók olyan címeken, amelyek kettővel oszthatók, a szók címeinek néggyel oszthatónak kell lenniük és a kettősszók címeinek pedig nyolccal oszthatónak kell lenniük. A formátumok határelrendezéseit a 9.2. ábrán illusztráljuk. A rögzített hosszúságú formátumon belül a formátumot alkotó biteket balról jobbra, egymás után következően számozzák, nullánál kezdve a számozást.

Ha explicit módon meghatározzák a mező hosszúságát, akkor változó hosszúságról beszélünk. Amint a 9.2. ábrán látható, a decimális szám vagy a karakterlánc (string) változó hosszúságú. A változó hosszúságú mezők 1-től 256 byte-ig terjedhetnek, egy byte-onként növekedve; eközben nem kell a határokra tekintettel lenni. A tömörített (packed) decimális formátumú decimális számok 1...16 byte-ig terjedhetnek.

9.2. ábra
Adatformátumok (IBM)

9.2.2. Adatábrázolások

Az IBM System/360 rendszerben öt adattípus van:

— fixpontos szám;
— lebegőpontos szám;
— decimális szám;
— logikai operandus és
— karakterlánc (string).

A fixpontos számok előjeles 2-es komplemens ábrázolású bináris egész számok. Amint a 9.2. ábrán látható, a fixpontos számok félszót vagy teljes szót foglalhatnak el; előjelbitjük a balról legelső bitpozícióban van.

A lebegőpontos számok hexadecimális számok, amelyek 16 számjegyszimbólumot használnak (9.1. táblázat). A lebegőpontos szám szó vagy kettősszó hosszúságú lehet.

9.1. táblázat
Hexadecimális kód

Bináris	Hexadecimális
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

9.2. táblázat
Tömörített decimális számok előjelének ábrázolása

Bináris szám	Értelmezés
1010	+ előjel az ASCII-8-ban
1011	— előjel az ASCII-8-ban
1100	+ előjel az EBCDIC-ben
1101	— előjel az EBCDIC-ben
1110	+ előjel bármely kódhoz
1111	— előjel bármely kódhoz

Amint a 9.2. ábrán látható, a balról legelső bit az előjelbit, a következő hét bit a kitevő és a többi a törtrész (törtrészbitek), amelyek vagy hat, vagy tizennégy hexadecimális számjegyet (24 vagy 56 bit) alkotnak. A törtrész előjeles ábrázolású. A kitevő 7-bites bináris szám, amelynek alapszáma 16 és decimális 64 hozzáadásával el van tolva. Ily módon az eredő kitevő mindig egész szám. A decimális számok nagyságrendje, amelyet lebegőpontos alakban lehet kifejezni, 16^{-64}-től 16^{+63}-ig terjed, ami közelítőleg 10^{-78} és 10^{+75} között van. A nulla törtrészű lebegőpontos számhoz automatikusan nulla kitevőt rendelnek hozzá.

A decimális számokat 4-bites bináris számokkal úgy kódolják, hogy két decimális számjegy van minden byte-ban. Ezt a formátumot *tömörített decimális formátumnak* nevezik. Használatos egy másik formátum is, amelyet *zónázott decimális formátumnak* neveznek, ezt a következőkben ismertetjük. Mindkét formátum a 9.2. ábrán látható. A tömörített decimális formátumú decimális számok változó hosszúságúak, 1-től 16

byte-ig. Ábrázolásuk előjeles egész szám, ahol az előjel a legkisebb helyiértékű bitben van. Csak a 0000-tól 1001-ig terjedő bináris számok (decimális 0...9) az érvényes decimális számjegy kódok. Az 1010-től 1111-ig terjedő kódokat az előjel ábrázolására használják. A 4-bites előjel értelmezését mutatja a 9.2. táblázat; az ASCII—8 és az EBCDIC kódokat a későbbiekben ismtertetjük.

A rögzített hosszúságú logikai operandusok egy, négy vagy nyolc byte-ból állnak. 4-byte-os rögzített hosszúságú logikai operandus látható a 9.2. ábrán. A karakterlánc (string) 1-től 256 byte-ig terjedő láncot alkot.

9.2.3. Adatkódok

A legismertebb adatkódok egyike a *binárisan kódolt decimális kód* (**BCD**). A BCD kód hat bitet alkalmaz 64 alfanumerikus és speciális karakterhez (9.3. táblázat). A táblázatban az első oszlop a karaktereket, a harmadik oszlop a BCD kódokat tartalmazza. Ez utóbbiakat külső BCD kódnak nevezik. A második oszlop más típusú BCD kódot mutat, amelyet belső BCD kódnak neveznek. Ezeket egyes számítógépekben — pl. az IBM 7090 számítógépcsaládban — akkor használják, amikor karaktereket tárolnak

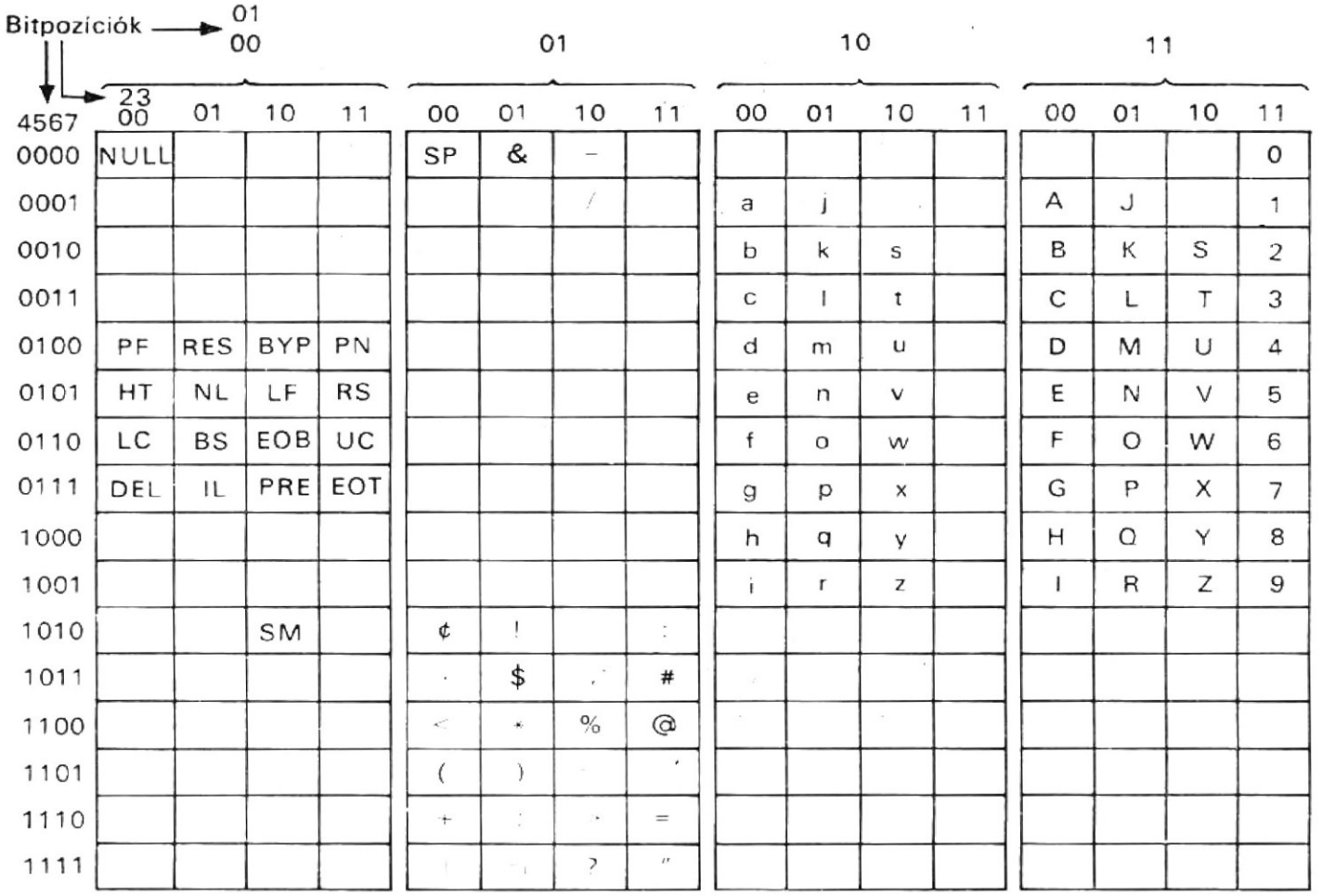
9.3. táblázat

Belső és külső BCD kódok

Karakter	Belső BCD kód	Külső BCD kód
0	0	12
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	10	10
9	11	11
A	21	61
B	22	62
C	23	63
D	24	64
E	25	65
F	26	66
G	27	67
H	30	70
I	31	71
J	41	41
K	42	42
L	43	43
M	44	44
N	45	45

9.3. Táblázat folytatása

Karakter	Belső BCD kód	Külső BCD kód
O	46	46
P	47	47
Q	50	50
R	51	51
S	62	22
T	63	23
U	64	24
V	65	25
W	66	26
X	67	27
Y	70	30
Z	71	31
+	20	60
—	40	40
*	54	54
/	61	21
	60	20
$	53	53
.	33	73
)	34	74
(	74	34
=	13	13
'	73	33



PF	Ilyukasztó ki	BS	Visszalépés	SM	Üzemmód változtatás
HT	horizontális tabulálás	IL	Üres futás	PN	Lyukasztó be
LC	Kisbetű	BYP	Elkerulés (bypass), különleges sorozat eleje	RS	Olvasó állj
DEL	Törölni	LF	Soremelés	UC	Nagybetűk
RES	Visszatárolás (restore)	EOB	Blokk vége	EOT	Átvitel vége
NL	Új sor	PRE	A két következő sorozat jelentése változik	SP	Szóköz

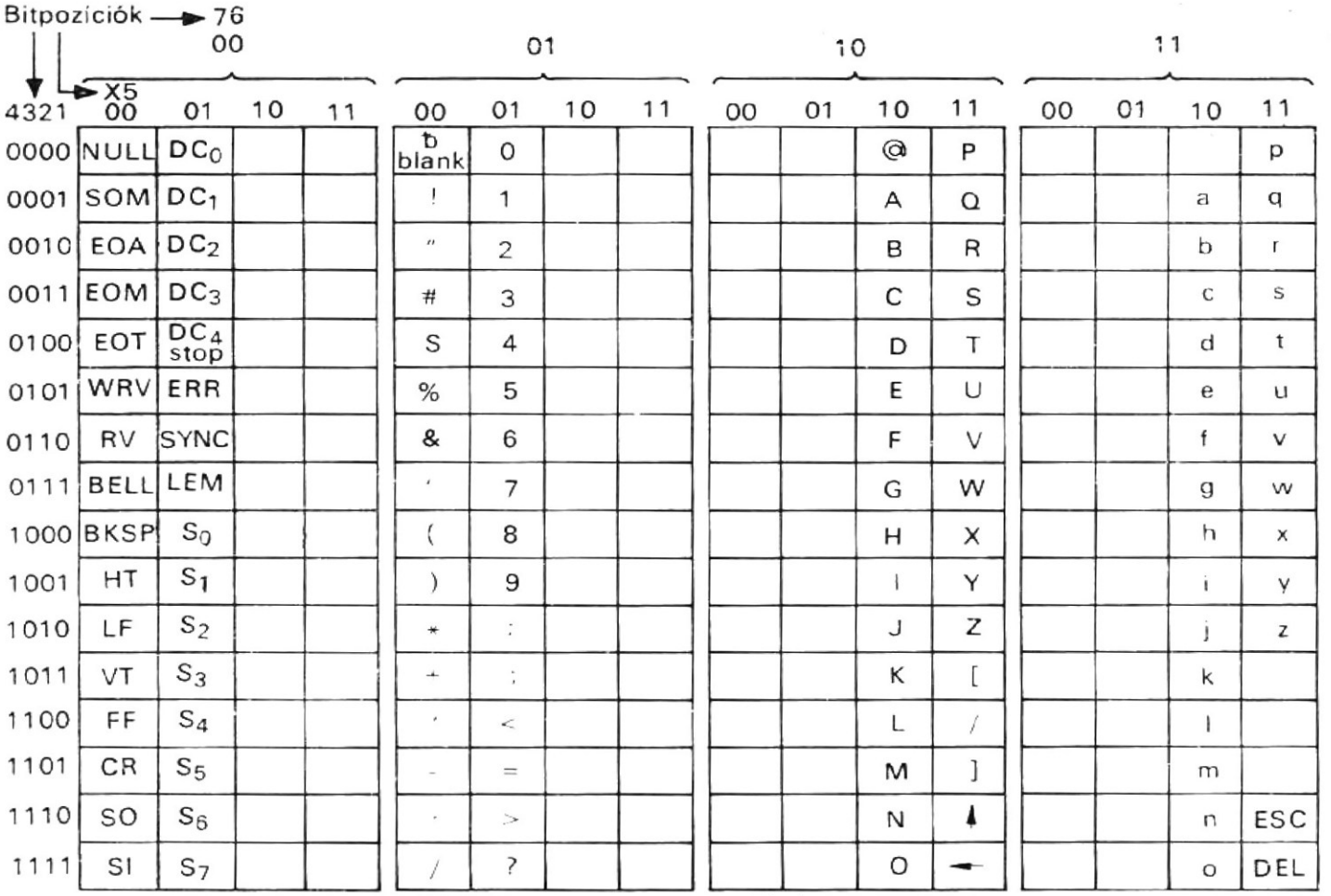
Bitpozíciók	0	1	2	3	4	5	6	7

9.3a ábra
A kiterjesztett binárisan kódolt decimális cserekód (EBCDIC)—(IBM)

a főtárban. A BCD kód széles körű elterjedtségének egyik oka, hogy egyszerűen fordítható az IBM kártyakódról.

A 6-bites kóddal nem lehet kielégítően reprezentálni 64 különböző karakternél többet. 8 bit 256 karakter vagy két binárisan kódolt decimális számjegy kódolását teszi lehetővé. A 6 bitről 8 bitre kiterjesztett BCD kódot választották az IBM System/360 számítógépeihez. Ezt a kiterjesztett változatot *kiterjesztett binárisan kódolt decimális cserekódnak* (**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode: EBCDIC) nevezik (9.3*a* ábra). Figyeljük meg a 9.3*a* ábrán látható bitpozíciók számozását. Azt, hogy éppen nyolc bitet választottak byte-nak, nagymértékben befolyásolta a 8-bites kód elfogadása. A lyukkártyán alkalmazott Hollerith-kódról EBCDIC-kódra történő konvertálást a 9.4. táblázat mutatja.

Ugyancsak az IBM System/360 számítógépekhez választott másik karakterkód az *információcsere amerikai standard kódjának* (**A**merican **S**tandard **C**ode's **I**nformation **I**nterchange: ASCII) 7-bites karakterkódja. Ezt a kódot terjesztették ki 8-bites kóddá, ezzel alakult ki az ASCII—8 kód (9.3*b* ábra).



9.3b ábra
Az amerikai standard információcsere kód 8-bites kiterjesztése (ASCII-8) — (IBM)

Ha egy decimális számot EBCDIC kóddal ábrázolnak, akkor azt *zónázott decimális formátumnak* nevezik (9.2. ábra). Ebben a formátumban a négy első bit — amelyet zónarésznek is neveznek — csupa 1-esből áll. A vezető 1-eseket úgy választják meg, hogy olyan egyeztetett, összeillő sorozatot alkossanak, amelyben a számok magasabban vannak, mint a betűk az alfanumerikus mezőkben. A zónázott formátumot elsődlegesen a karakterérzékelő be-, ill. kimeneti készülékeihez használják. A decimális aritmetikai műveleteket decimális számokon tömörített decimális formátumban végzik (nem zónázott decimális formátumban). A gépcsaládban rendelkezésre áll egy utasítás, amely a decimális számokat zónázott formátumból tömörített formátumra konvertálja. Egy másik utasítás szolgál a tömörített decimális formátumú decimális számok bináris számokká alakítására, a bináris aritmetikai műveletekhez.

9.4. táblázat

Hollerithkártya kód és EBCDIC kód

Lyukkártyakód	Nyomtatókarakter	EBDCIC kód
	üres	01000000
12;8;3	. (tizedespont)	01001011
12;8;4	<	01001100
12;8;5	(	01001101
12;8;6	+	01001110
12	&	01010000
11;8;3	$	01011011
11;8;4	*	01011100
11;8;5	)	01011101
11	–	01100000
0;1	/	01100001
0;8,3	,	01101011
0;8,4	%	01101100
8;3	#	01111011
8;4	@	01111100
8;5	'	01111101
8;6	=	01111110
12;1	A	11000001
12;2	B	11000010
12;3	C	11000011
12;4	D	11000100
12;5	E	11000101
12;6	F	11000110
12;7	G	11000111
12;8	H	11001000
12;9	I	11001001
11;1	J	11010001
11;2	K	11010010
11;3	L	11010011
11;4	M	11010100
11;5	N	11010101
11;6	O	11010110
11;7	P	11010111
11;8	Q	11011000
11;9	R	11011001
0,2	S	11100010
0,3	T	11100011
0,4	U	11100100
0,5	V	11100101
0;6	W	11100101
0;7	X	11100111

9.4. táblázat folytatása

Lyukkártyakód	Nyomtatókarakter	EBDCIC kód
0;8	Y	11101000
0;9	Z	11101001
0	0	11110000
1	1	11110001
2	2	11110010
3	3	11110011
4	4	11110100
5	5	11110101
6	6	11110110
7	7	11110111
8	8	11111000
9	9	11111001

9.2.4. Utasításformátumok

A 9.4. ábrán öt alapvető utasításformátumot láthatunk. Ezek az utasításformátumok egy, két vagy három félszót tartalmaznak. Minden utasítást a tárban levő félszó határokhoz kell igazítani.

Az öt bázisutasítás-formátumot a következőképpen jelölik:

RR, RX, RS, SI és SS.

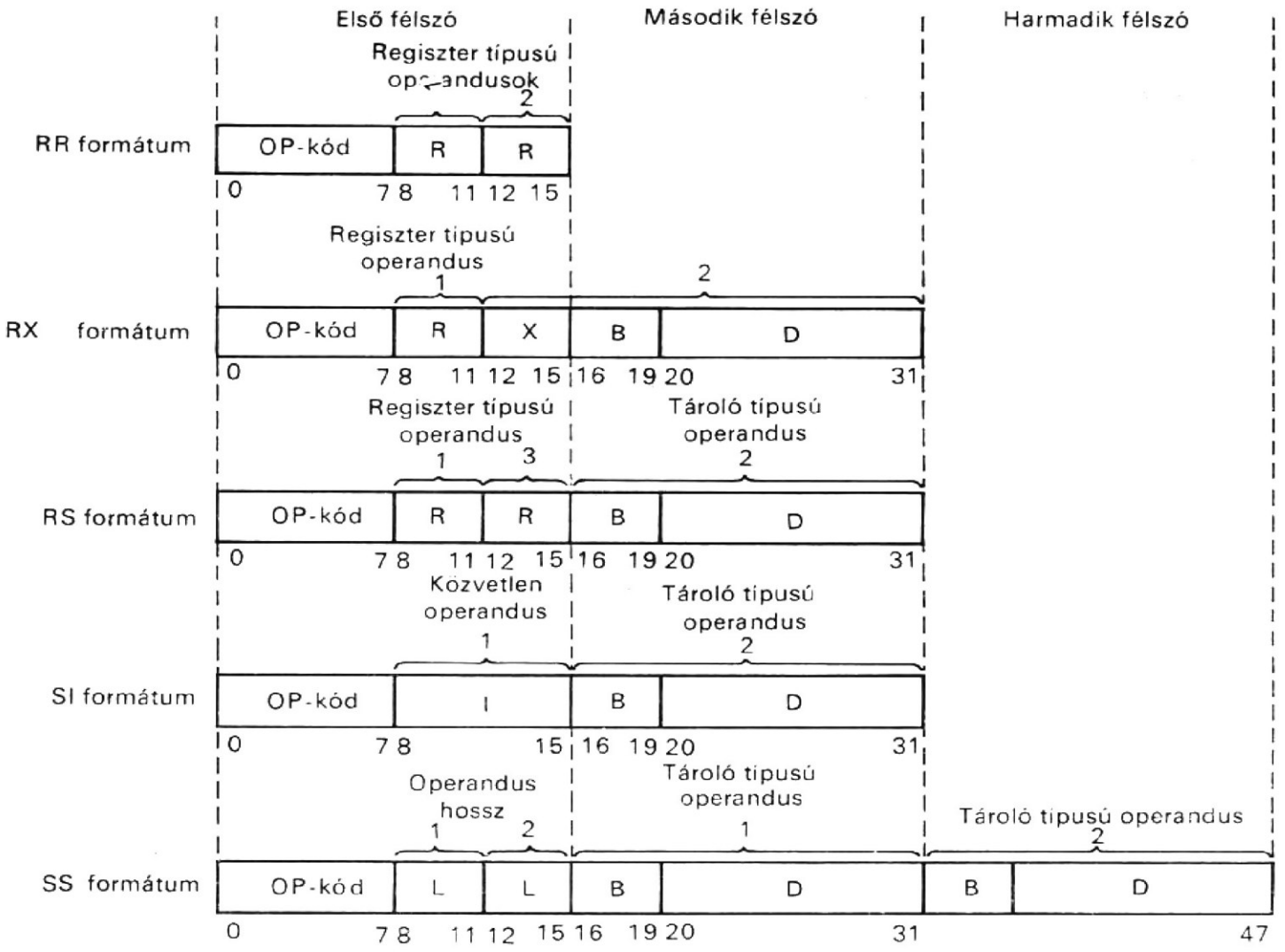
Az utasítás első nyolc bit hosszúságú mezője tartalmazza a műveleti kódot (op-kód). Az op-kód a következőképpen van kódolva. A 0 és 1 bit az utasítás hosszát specifikálja, amint a 9.5. táblázatban látható, és közvetve ezek a bitek határozzák meg az adat helyét is (főtár vagy regiszter). A 2. és 3. bit az adat típusát (fix vagy változó hosszúságú, decimális, bináris vagy lebegőpontos) specifikálja. A 4...7. bitek azt specifikálják, hogy mit kell az adatokkal tenni. Az utasítás többi mezője az operandusokat specifikálja. Az operandusoknak négy típusa van:

1. regiszteroperandus;
2. tároperandus;
3. közvetlen operandus;
4. blokkoperandus.

9.5. táblázat

Az utasításhossz kódja

Formátum	A műveleti kód első két baloldali bitje	A félszavak száma
RR	00	1
RX	01	2
RS vagy SI	10	2
SS	11	3



9.4. ábra
Utasításformátumok (IBM)

A regiszteroperandust a 16 általános rendeltetésű regiszter egyike tartalmazza. Ezek 32-bites regiszterek, és itt a továbbiakban egyszerűen regiszteréknek nevezzük őket. 4-bites című regisztereket R jelöli, amint az a 9.4. ábrán látható. Az RR formátum két regiszterben levő operanduson végzett műveletet jelöl, ahol az eredményt az első operandus helyére írják be.

A tároló típusú operandust a főtár egyik helyén tárolják. A tár címe 24 bit. Ezt a címet úgy határozzuk meg, hogy összeadjuk az egyik — B-vel címzett regiszterben levő 24-bites — báziscímet egy másik — X-szel címzett regiszterben tárolt 24-bites — indexcímet, valamint az eltolási címet — röviden eltolást (displacement) —, amely az utasításban levő 12-bites D-vel jelzett mező.

Ezt az összeget nevezik *tényleges címnek*. A tényleges cím kialakításakor a bázis- és indexcímet előjel nélküli, 24-bit hosszúságú egész számként kezelik, az eltolási címet pedig 12-bites pozitív, bináris egész számként, a túlcsordulást elhanyagolva. Ha akár a bázisrész, akár az indexrész nulla, akkor ezt a nullát kell használni a tényleges cím kialakításához. Az RX formátum olyan műveletet jelöl, ahol az egyik operandus a regiszterben, a másik a főtárban van.

A blokkoperandusok operanduscsoportra utalnak; ezek az általános rendeltetésű regiszterekben lehetnek. Az RS formátum egy általános regiszterhalmazban levő blokkoperandusokon végzett műveletet jelöl, az R_1-gyel specifikált regisztertől kezdve és az R_3-mal specifikált regiszterrel végezve. Az eredményt egy blokkban tárolják a tárban, amelyek kezdő címét a B_2-vel specifikált báziscím és a D_2 eltolás összeadásával határozzuk meg. Ez a formátum a tárban levő blokkoperandusokon végzett műveletet is jelölhet, itt az eredményt általános regiszterek tartalmazzák.

A közvetlen (immediate) operandus maga a 8-bites I mező. Ez az I mező olyan karakter, amelyet közvetlenül lehet felhasználni. Az SI formátum közvetlen operanduson végzett műveletet jelöl, ahol az eredmény olyan tárhelyre kerül, amelynek címét a B által meghatározott báziscím és a D eltolás összegeként kapjuk meg.

Blokkoperandusok lehetnek a főtárban is. Ezeket egy tárcím és byte-számként megadott operandushossz segítségével lehet címezni. Az SS formátum két, a főtárban levő, változó hosszúságú blokkoperanduson végzett műveletet jelöl. A két tárcím mindegyikét a B által meghatározott báziscím és a D eltolás összegeként kapjuk meg, és mindegyik operandus hosszát az utasítás L mezője specifikálja.

A bázis—eltolás címzés alkalmazásának két előnye van. Az első előnye az, hogy redukálni lehet a címzéshez szükséges utasításbiteket; ez a bitmegtakarítás teszi viszont szükségessé az általános rendeltetésű regiszterek alkalmazását a tár címzéséhez. A másik előny az, hogy egyszerűbb programáthelyezést tesz lehetővé, mivel a báziscímeket a betöltéskor lehet specifikálni.

9.3. Számítógép-szervezés

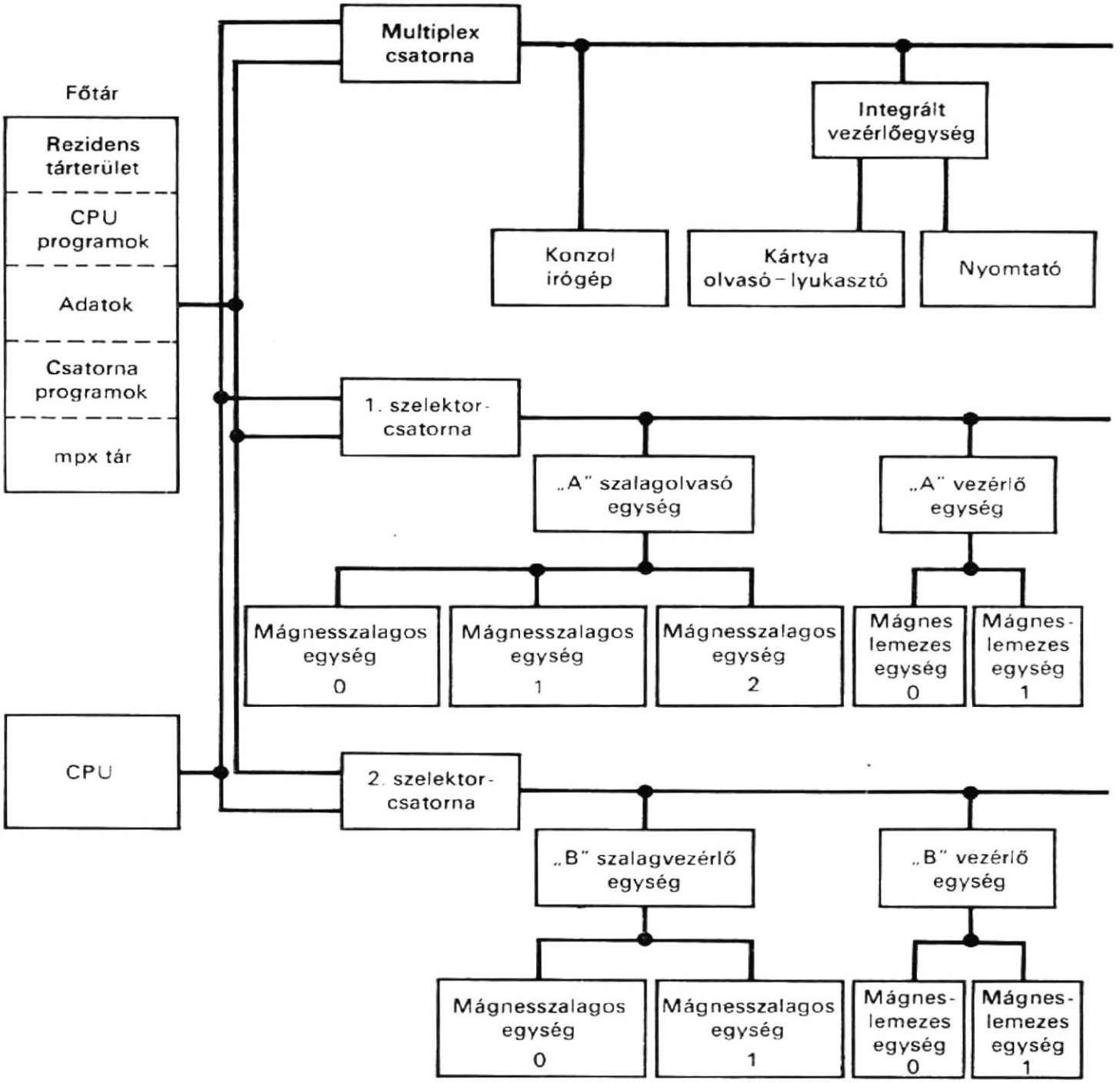
Azért, hogy az IBM System/360 számítógépcsaládot e fejezetben belül részletesebben ismertetni tudjuk, a leírást csak a 40-es modellre korlátozzuk. Bár ez a családnak egy közepes nagyságú tagja, a fogalmak és a konfigurációk általánosan alkalmazhatók a család többi tagjára is.

IBM System/360 40-es modelljén alapuló számítógéprendszert mutatunk be a 9.5. ábrán. Ez a modell mikroprogramozott számítógép. Van egy főtára, egy központi feldolgozóegysége (CPU), egy multiplexcsatornája és két szelektorcsatornája; az utóbbiak opcionálisak. A csatornák a be-, ill. kimeneti készülékekhez a családnál szabványosított B/K illesztésen (standard interface-en) keresztül kapcsolódnak. A multiplexcsatornához egy konzolírógép, valamint — integrált vezérlőegységen keresztül — egy sornyomtató és egy lyukkártyaolvasó—lyukasztó csatlakozik. Három mágnesszalagegység kapcsolódik az 1-ső szelektorcsatornához az A mágnesszalagvezérlő egységen keresztül és két mágneslemezegység az A jelű vezérlőegységen keresztül. Két mágnesszalag csatlakozik a 2-ik szelektorcsatornához a B mágnesszalagvezérlő egységen keresztül és két mágneslemezegység a B jelű vezérlőegységen keresztül.

9.3.1. Adatátvitel

Az adatátvitel a tárak, regiszterek, adatutak és egyéb funkcionális elemek közötti kapcsolat, amely az adatok és vezérlések átvitelének lehetséges módjait mutatja és lényeges szerepe van a különböző mikroműveletek és műveletsorozatok végrehajtásában.

Az **adatok:** címek, számok, kódok, jelzőbitek (flag), számlálóértékek, parancsok, indikátorok, állapotszavak, vezérlőszavak, idő, utasítások, karakterláncok stb. lehetnek. Az adatátviteli módok jellemzik a CPU és a tárak közötti kapcsolatokat. A 40-es modell adatátviteli útjait a 9.6. ábra blokkdiagramja mutatja. Amint látható, a lényegesebb adatutak a következők: az R regiszterhez kapcsolódó sín, az aritmetikai és logikai egység (ALU) két bemeneti sínje, az ALU kimeneti sínje és a B/K készülékekkel történő átvitelhez kapcsolódó illesztések.



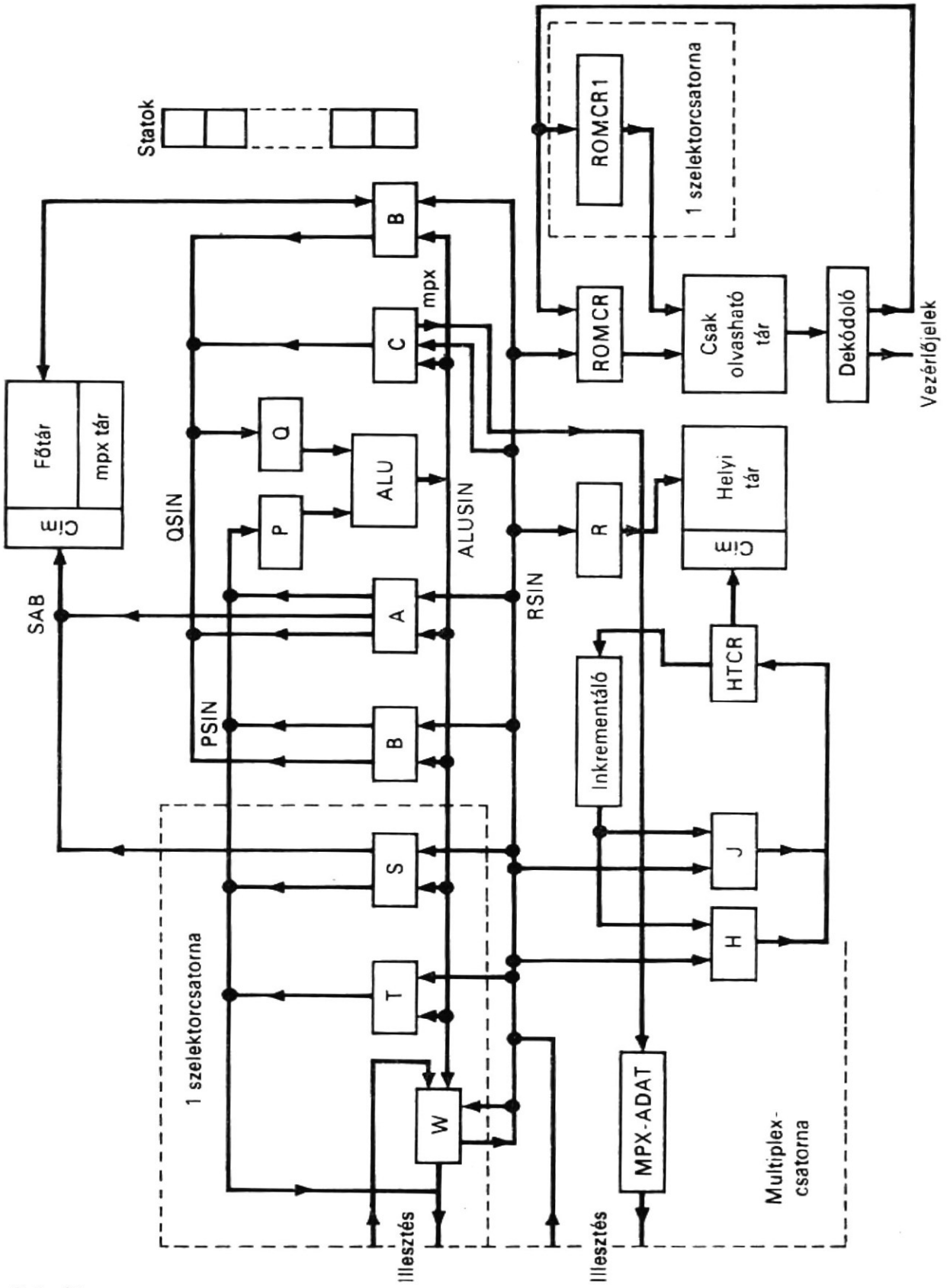
9.5. ábra
Az IBM System/360 40-es modellje

9.3.2. Tárak

A gépben négy tár van:

- a főtár;
- multiplextár;
- helyi (lokális) tár;
- csak olvasható tár.

A főtár a programot, adatokat és a vezérlőszavak egy részét tárolja. A multiplextár a multiplexcsatorna vezérlőszavait tárolja. A helyi tár általános rendeltetésű regisztereket és átmeneti tárhelyet biztosít a csatorna működéséhez. A csak olvasható tárban (**R**ead **O**nly **M**emory: ROM) a mikroprogram helyezkedik el. E tárak jellemző paramétereit a 9.6. táblázat foglalja össze. Magukat a tárakat a következőkben mutatjuk be.



9.6. ábra
Egyszerűsített adatutak az IBM System/360 40-es modellben

9.6. táblázat

Az IBM System/360 40-es modelljében levő tárak főbb jellemzői

Jellemzők	Főtár	Multiplextár	Helyi tár	Csak olvasható tár
Kapacitás*	16K—256K byte	128...1024 szó	144 szó	4K, 6K vagy 8K
Ciklusidő	2,5 μs	2,5 μs	1,25 μs	625 ns
Szóhossz	2 byte vagy 16 bit	2 byte vagy 16 bit	22 bit	56 bit
A cím hossza	14...18 bit	7...10 bit**	8 bit	12 vagy 13 bit

* K 1024-szeres szorzót jelent.
** Az Y1 stattal együtt.

9.3.2.1. Főtár

A főtár közvetlen hozzáférésű, mágnesmagostár 2,5 μs ciklusidővel. A főtár elérése az olvasási vagy írási műveletben teljesen független; ez *osztott olvasás—írás ciklusként* (split cycle) ismeretes. Mind az olvasási, mind az írási ciklusban két byte-ot (18 bitet, a két paritásbittel együtt) visznek át egyszerre. Ennek időtartama 1,4 μs az olvasási és 1,1 μs az írási ciklusban. Az osztott ciklusú működésben az írási ciklust mindig megelőzi egy olvasási ciklus. A főtár kapacitása 16 K, 32 K, 64 K, 128 K, ill. 256 Kbyte (ahol 1 Kbyte jelentése 1024 byte) és 14...18 bittel címezhető. A maximális méretű főtárat két tárolóegységbe szervezik, ahol mindegyik egységben 131 072 byte van és két tömböt tartalmaz (mindegyik tömb maximum 64 Kbyte-ot tárol). A tárhoz tartozik a D tartalom- (puffer- vagy adat-) regiszter, amelyet a tárcímsínen (TCS) megjelenő cím címez meg. A TCS az A regiszterhez csatlakozik, ha a feldolgozóegység (vagy a multiplexcsatorna) van működésben, míg ha az 1. szelektorcsatorna működik, akkor az S1 regiszterhez, ha pedig a 2. szelektorcsatorna, akkor az S2 regiszterhez. A főtár a CPU-tól vagy a be-, ill. kimeneti egységektől jövő adatokat tárolja, ill. ezek számára szolgáltat adatokat. Minthogy egy 256 Kbyte-os főtár címzéséhez 18 bitre van szükség, a címregiszternek két byte-nál nagyobbnak kell lennie; azt a külön három bitet, amely erre és egyéb célokra is szolgál, *kiegészítő* (extender) *bitnek* nevezik.

131 072 byte kapacitású főtárat feltételezve az FT főtár és a hozzátartozó regiszterek és sínek a következőképpen írhatók le:

```
Comment, a főtár leírása                                        (9.1)
Register,    A(0—21),          $FT címregiszter
             D(0—17),          $FT tartalom- (puffer-) regiszter
Bus,         TCS(0—21),        $tárcímsín
Memory,      FT(TCS)=FT(0—131 071, 0—17),
```

9.3.2.2. Multiplextár

A multiplextár (MPX) a főtár járulékos helyeit foglalja magában. Külön Y meghajtóvonalai és címsíne van; egyébként vezérlése, ütemezése a főtárral közös, és közösek egyéb áramkörei. Kapacitása 256-tól 2048 byte-ig változhat. Kapacitását a főtár kapacitásához viszonyítva lehet megválasztani. Az MPXCS multiplex címsínnel címezhető. 2048 byte kapacitású multiplextárat feltételezve, a következőképpen írható le:

```
Comment, a multiplextár leírása                                    (9.2)
Bus,        MPXCS(15—13, 6—0),
Memory,     MPX(MPXCS)=MPX(0—1023, 0—17),
```

A multiplextárhoz a hozzáférés egy speciális vezérlőbit, az mpx stat-nak nevezett Y1 bit beállításával történik. Ezt a későbbiekben mutatjuk be. Ehhez csak a mikroprogram tud hozzáférni. A multiplex-tárolóegység vezérlőszavakat és egyéb vezérlőinformációt tartalmaz a multiplexcsatornák működtetéséhez.

9.3.2.3. Helyi tár

A helyi tár, a HT, nagysebességű mágnesmagostár; a kapacitása 144 22-bites szó. A tár osztott ciklusban működik. Egy teljes olvasás vagy írás ciklus 1,25 μs. Az elérési idő 350 ns. Az R a tartalomregiszter, a HTCR a címregiszter. A H és J regiszterek tárolják a HT címét, továbbá, működésének leírására bevezetünk egy inkrement operátort. Ezeket a regisztereket és a helyi tárat az alábbi utasítások írják le.

```
Comment, a helyi tár leírása                                       (9.3)
Register,   HTCR(0—7),          $a HT címregisztere
            H(0—7, P),          $a HT címének tárolása
            J(0—7, P),          $a HT címének tárolása
            R(0—21),            $a HT tartalomregisztere
Memory,     HT(HTCR)=HT(0—143, 0—21),
Operator,   X(4—7)←Y(4—7) inkrement k,
```

A fenti inkrement operátor 4-bit inkrementálását deklarálja, ahol a k: +1, 0, −1 vagy −2 lehet. A HTCR regiszter, az inkrement operátor a H és J regiszterekkel együttesen alkotja a „címhurkot". A címhurok segítségével a HTCR regiszter tartalma az inkrementálón keresztül jut el a H és J regiszterekhez, és ezek a regiszterek tárolják a HT címét, amely majd a HTCR-ben kerül felhasználásra.

A helyi tárhoz tartozó tárolótérképet láthatunk a 9.7. táblázatban. Ebből kitűnik, hogy a HT helyi tár — egyebek között — 16 fixpontos regisztert, négy lebegőpontos regisztert, csatornapuffereket, tárat az egységvezérlőszó (EVSZ, ill. UCW)részére, valamint a „mozgatási" (dump) területek első és második szintjét tartalmazza. A gépi nyelv szintű program számára csak a fixpontos és lebegőpontos regiszterek érhetők el. A helyi tár alkalmazása teszi lehetővé, hogy a CPU a csatornaműveletek szempontjából időosztásos üzemben működjék, azáltal, hogy átmenetileg tárolja a CPU regisztereinek tartalmát. Ezért nincs szükség teljességükben implementált csatornákra.

9.3.2.4. Csak olvasható tár

A csak olvasható tár, a ROM, 16 modulból áll, amelyek egyenként 256 56-bites szót tartalmaznak. A teljes tár kapacitása 4096 szó. A ROM ciklusideje és a CPU ciklusideje egyaránt 0,625 μs. Tehát a főtár ciklusának minden 1 μs-ára négy, 0,625 μs-os ROM ciklusidő jut. A ROMCR címregiszter címzi meg a csak olvasható tárat a CPU vagy a multiplexcsatorna működése közben. Az 1-es szelektorcsatorna működésekor a ROMCR1, a 2-es szelektorcsatorna működésekor pedig a ROMCR2 regiszter címe-

9.7. táblázat

A helyi tár felosztása a 40-es modellben

Hely	Rendeltetés
000—008	Munkaterület
009—015	Munkaterület és „logout" (adatkiviteli) terület
016—031	Nincs felhasználva
032—037	UCW (EVSZ) az 1. szelektorcsatorna részére
038—041	A multiplexer-csatorna munkaterülete
042	Megszakításpuffer
043—047	Kijelöletlen
048—053	UCW (EVSZ) a 2. szelektorcsatorna részére
054—056	Kijelöletlen
057—063	Második szintű „töltési" (dump) terület
064—066	Munkaterület meghatározatlan állapotban
067—071	Programállapotszó
072	Start I/O kapcsoló
073—079	Első szintű „töltési" (dump) terület
080—095	Nincs felhasználva
096—111	Puffer az 1. szelektorcsatorna részére
112—127	Puffer a 2. szelektorcsatorna részére
128—191	Nincs felhasználva
192—207	Lebegőpontos regiszterek
208—223	Nincs felhasználva
224—255	Fixpontos regiszterek

zi meg a csak olvasható tárat. A tárat és a hozzá tartozó regisztereket az alábbiakban írjuk le.

Comment, a csak olvasható tár leírása (9.4)

```
Register,   ROMCR(0—11),          $ROM tár címregisztere
            ROMCR1(0—12, P),      $ROM tár 1. szelektorcsatorna címregisztere
            ROMCR2(0—12, P),      $ROM tár 2. szelektorcsatorna címregisztere
Memory,     ROM(ROMCR)=ROM(0—255, 0—55),
```

9.3.3. Regiszterek

A 40-es modellben levő regiszterek a következő csoportokból állnak:

— adatregiszterek;
— ALU regiszterek;
— csatornaregiszterek;
— ROM címregiszterek;
— a tárvédelem regiszterei és
— a stat-ok.

Az IBM terminológiájában valamely regisztert adatsínnek lehet tekinteni, ha minden órajelciklusban törlik, vagyis visszaállítják 0-ra. Az R, P, Q és a többi, a következőkben leírásra kerülő regiszterre úgy is lehet hivatkozni, mint adatsínekre.

9.3.3.1.
Adatregiszterek

Öt adatregiszter van: A, B, C, D és R. Ezek közül mindegyik két 9-bites részregisztert (amelyben nyolc adatbit és egy paritásbit van) és egy 4-bites részregisztert (három adatbittel és egy paritásbittel) tartalmaz. Ezeket a regisztereket az alábbiakban írjuk le.

```
Comment, az öt adatregiszter leírása                                    (9.5)
Register,    A(0—21),        $a főtár címregisztere
             B(0—17),        $átmeneti tároló
             C(0—21),        $átmeneti tároló
             D(0—17),        $a főtár tartalom- (puffer-) regisztere
             R(0—21),        $pufferregiszter a helyi tár részére, adatok
                               és a kijelzés tárolására
Subregister, A0(0—7, P)=(0—8),
             A1(0—7, P)=A(9—17),
             AX(5—7, P)=A(18—21),
```

A fenti, AX részregisztert extenzióregiszternek nevezik. A C és R regisztereket az A-hoz hasonlóan definiálják. A B részregisztereit az alábbiakban határozzuk meg.

```
Subregister, B0(0—7, P)=B(0—8),
             B1(0—7, P)=B(9—17),
```

A D részregisztereit hasonlóképpen definiálják.

A felsorolt regiszterek között adatút van (l. 9.6. ábra). Az A, B, C vagy D regiszterekbe akár az R regiszterből, akár az ALU-ból (aritmetikai és logikai egység) be lehet írni és e regiszterek bármelyikéből adatokat lehet átvinni a Q regiszterbe. Az A regiszterben levő adatokat ezen kívül még a P regiszterbe és a TCS (tárcím) sínhez is át lehet vinni, míg a B-ben levő információt a P regiszterbe lehet átvinni. Az R regiszterben levő adatokat az A, B, C vagy D regiszterekbe lehet átvinni.

9.3.3.2.
ALU regiszterek

Hét ALU regiszter van. Ezeket az alábbiakban írjuk le.

```
Comment, a hét ALU regiszter leírása                                    (9.6)
Register,    P(0—7, P),              $operandusregiszter
             Q(0—7, P),              $operandusregiszter
             CONTROL(0—3, P),        $ALU funkcióregiszter
             FUNKTION(0—4, P),       $ALU funkcióregiszter
             EXTENZIÓ(0—2, P),       $extenzióregiszter
             SKEW-SELECT(0—7, P),    $az egyik oldal kiválasztásához tartozó
                                       regiszter
             SKEW-BUFFER(0—3, P),    $az egyik oldal tartalom- (puffer-) re-
                                       gisztere
```

A P és Q regiszter az ALU bemenetén levő két regiszter a két operandus számára. A CONTROL regiszter a végrehajtandó ALU funkciót határozza meg. A FUNKTION

regiszter az indirekt műveletekhez szolgáló ALU funkciót tárolja. Az EXTENZIÓ regisztert bemeneti pufferként használják az egyik extenziós bithez. Az AO és AX (vagy CO és CX, vagy RO és RX) részregiszterekben levő adatokat a P vagy Q regiszterekbe írják át. Amikor a Q regiszterből a SKEW-BUFFER regiszteren keresztül adatokat visznek át, 4-bites balra léptetést lehet előírni.

9.3.3.3. Csatornaregiszterek

Két multiplexcsatorna-regiszter van:
— az MPX-ADAT;
— az MPX-HIBA.

Az MPX-ADAT regiszter a B/K illesztéshez továbbítandó adatbyte átmeneti tárolását végző regiszter. Az MPX-HIBA regiszter hat egybites részregiszterből áll, amelyeket a multiplexcsatorna működése közben észlelt különböző állapotok állítanak be. Ezeket az alábbiakban írjuk le:

```
Comment, multiplexcsatorna-regiszterek                          (9.7)
Register,   MPX-ADAT(0—7, P),     $interface pufferregisztere
            MPX-HIBA(0—5),        $hat hiba-indikátora
```

A szelektorcsatorna regisztereit egy későbbi fejezetben írjuk le.

9.3.3.4. A tárvédelem regiszterei

A tárvédelemnek két regisztere van:
— a TV-ADAT;
— a TV-KULCS.

Ezek a 9.6. ábrán nem láthatók. A TV-ADAT tárvédelem-adatregiszter tartalmazza a tárvédelmi kulcsot akkor, amikor hozzáférés történik a főtárhoz, míg a TV-KULCS tárvédelem kulcsregiszter a tárvédelmi kulcsot tartalmazza a CPU és a multiplexcsatorna műveletek esetén. Ezeket az alábbiakban írjuk le.

```
Comment, a tárvédelem regiszterei                               (9.8)
Register,   TV-ADAT(P, 0—3),      $tárvédelem-adatregiszter
            TV-KULCS(P, 0—3),     $tárvédelem-kulcsregiszter
```

9.3.4. A statok

A statok (IBM terminológia) egybites regiszterek aggregátumai, amelyek gépállapot és ellenőrzési feltételek jelzésére szolgálnak. Két kategóriára osztják őket: Y-statok és nem-Y-statok. Kilenc nem-Y-stat van, ezeket az alábbiakban írjuk le.

```
Comment, a nem-Y-statok leírása                                 (9.9)
Register,   HALT,     $azt jelzi, hogy a számítógép stop hurokban van
            VÁRJ,     $azt jelzi, hogy a rendszer várakozási hurokban van
```

	MEHET,	$géphibát jelez „logout"-ot vagy hardware-hibát (hard-stop) jelez
	ASCII,	$ha értéke 1, a CPU ASCII-módban működik
	CPU,	$azt jelzi, hogy a CPU csatornaként működik-e, vagy sem
	VTC,	$védett tárhely címzését jelzi
	ÉTC,	$érvénytelen tárhely címét jelzi
	YCD,	$az ALU átvitelregisztere
	YCI,	$az ALU átvitelregisztere

A fentiekben szereplő „*logout*" (IBM terminológia) azt jelenti, hogy a CPU és a csatornavezérlő regiszterek tartalma a főtárba a hexadecimális 80 helytől kezdve átkerül.

16 Y-stat van, amelyek négy regisztersort alkotnak. Ezek neve YA, YB, YD és YE. A statokat az alábbiakban írjuk le.

```
Comment, az Y-statok leírása                                    (9.10)
Register,     Y0,          $tár-stat
              Y1,          $mpx stat
              Y2,          $feltételkód tárolása
              Y3,          $feltételkód tárolása
              P1,          $paritásbit
Casregister,  YA(0—4)=Y0—Y1—Y2—Y3—P1,

Register,     Y4,          $általános célú bit
              Y5,          $általános célú bit
              Y6,          $általános célú bit
              Y7,          $általános célú bit
              P2,          $paritásbit
Casregister,  YB(0—4)=Y4—Y5—Y6—Y7—P2,

Register,     Y8,          $„töltést" gátló stat (KG)
              Y9,          $maszkolható interrupt (MI)
              Y10,         $kézi stat
              Y11,         $nem használják
Casregister,  YF(0—3)=Y8—Y9—Y10—Y11,

Register,     Y12,         $hibastat
              Y13,         $nem használják
              Y14,         $integrált zérusteszt (IZT)
              Y15,         $betöltésstat
Casregister,  YE(0—3)=Y12—Y13—Y14—Y15,

Casregister,  Y(0—3, P1, 4—7, P2, 8—15),=YA—YB—YD—YE,
```

A fentiekben az Y0 társtat egyik állapotában a VTC és ÉTC statok ellenőrzését teszi lehetővé. Az Y1 multiplex az mpx stat, egyik állapotában a multiplextárhoz való hozzáférést engedélyezi. Az Y0, Y1, Y2, Y3 statok és a P1 paritás alkotják az YA regisztert. Az Y4, Y5, Y6, Y7 statok és a P2 paritás képezik az YB regisztersort. Az Y9 maszkolható megszakításstat bebillen, ha mpx megszakítás jön létre. Az Y10 kézi stat kézi betöltést, és a regiszterek és tár kijelzését teszi lehetővé. Amikor az Y12 hibastat be van kapcsolva, gépi hiba miatti leállást okoz, ha hiba fordul elő. Az Y15 betöltési stat akkor billen be, ha a betöltő nyomógombot lenyomják. Az YD és YE részregiszterek hasonlóképpen jönnek létre, de paritás nélkül. A 16 Y-stat alkotja az Y regisztersort.

9.3.5. Sínek

Négy sín (bus) vesz részt az adatátvitelben. A SINP és SINQ az ALU két bemeneti sínje, a P, ill. Q ALU regisztereken keresztül. Az A, B, S, T és W regiszterek a SINP-hez, míg az A, B, C és D regiszterek a SINQ-hoz kapcsolódnak. A SINALU sín az ALU kimeneti sínje. Ez az A, B, S, T és W regiszterekhez csatlakozik. Az R sín kapcsolja össze az A, B, C, D, S, T, W, ROMCR regisztereket és a multiplexcsatornától jövő bemeneti kapocspárokat. Ez három részre oszlik: SINR0, SINR1 és SINRX. A síneket az alábbiakban írjuk le.

Comment, a sínek leírása (9.11)

```
Bus,        SINP(0—7, P),                  $ALU P-sín
            SINQ(0—7, P),                  $ALU Q-sín
            SINALU(0—7, P),                $ALU kimeneti sín
            SINR0(0—7, P),                 $az R-sín első byte-ja
            SINR1(0—7, P),                 $az R-sín második byte-ja
            SINRX(5—7, P),                 $az R-sín extenziós bitje
Terminal,   SINR=SINRX—SINR0—SINR1,        $R-sín
```

9.3.6. Csatornák

A gépben van egy multiplexcsatorna (MCS), és opcionális lehetőségként két szelektorcsatorna (SCS). Az MCS-nek maximálisan 128 alcsatornája lehet és ezáltal 128, viszonylag kis sebességű be-, ill. kimeneti B/K készülék (pl. lyukkártyaolvasó vagy nyomtató) egyidejű működését teszi lehetővé. Ez a csatorna multiplex vagy burst-üzemmódban működhet. A multiplex üzemmódban egy be-, ill. kimeneti egység logikailag csak egy adatbyte átviteléhez szükséges ideig kapcsolódik a csatornához. Ezért a multiplex üzemmódot *byte-üzemmódnak* is nevezik. Az adatbyte-ok átvitele között az egység lekapcsolódik és a csatorna felszabadul, hogy egy másik egységhez vagy egységtől szállítson egyetlen byte-ot. A *burst-üzemmódban* ugyanaz a be-, ill. kimeneti készülék logikailag a csatornához kapcsolva marad a teljes be-, ill. kimeneti művelet befejezéséig.

Ha az MCS byte-üzemmódban működik, a felhasználható CPU idő fokozatosan csökken az MCS eredő adatátviteli sebességének növekedésével, egészen addig, amíg már semmilyen felhasználható idő sem áll rendelkezésre, ha az eredő adatátviteli sebesség eléri a 26 000 byte/s-ot. Ha az MCS burst-üzemmódban működik, valamennyi CPU erőforrás átadódik a csatornaművelethez; a CPU nem tud a be-, ill. kimeneti készülékkel egyidejűleg dolgozni. Ebben az üzemmódban a maximális adatátviteli sebesség 228 000 byte/s. Ezeket az értékeket a 9.8. táblázat mutatja.

A szelektorcsatornák a viszonylag nagy sebességű be-, ill. kimeneti készülékek adatátviteléhez használhatók. Minthogy itt mindössze egy alcsatorna van, csak egyetlen B/K vezérlőegységet lehet egyszerre az SCS-hez csatolni és az SCS csak burst-üzemmódban működik.

A szelektorcsatornáknak két típusa van: A és B típusú:

Az A típusú SCS nagy mértékű időosztásban működik a CPU regisztereivel és adatutaival, valamint a mikroprogram-vezérléssel. Az adatokat nem közvetlenül transzferálják a főtár és az illesztés között, hanem byte-onként, sorosan, a helyi tár-

ban levő 16-byte-os pufferen keresztül. Amikor a 16-byte-os puffer félig megtelik, az SCS kérést ad a CPU regiszterek és az adatutak használatára. A CPU regiszterek tartalmát átviszik, „kitöltik" a helyi tárba. Legfeljebb 12 csak olvasható tár (ROM) ciklusra van szükség ahhoz, hogy megőrizzék a CPU regiszterek pillanatnyi tartalmát és betöltsék a vezérlőszót. Ezután a pufferben levő byte-okat átviszik a főtárba, főtárciklusonként kétbyte per főtárciklus sebességgel.

9.8. táblázat

Maximális csatorna-adatátviteli sebesség a 40-es modellben (byte/s-ban)

Csatorna	Multiplex üzemben	Szelektorüzemben
Multiplexer	26 000	228 000
Szelektor egy	—	400 000
kettő	—	300 000 bármelyikhez

A központi feldolgozóegység regisztereinek átvitele (kitöltése) a multiplex- és a szelektorcsatornákba a következőképpen történik. Amikor az MCS-től megszakítás érkezik be, valamennyi, a pillanatnyi CPU művelet szempontjából lényeges adatot kitöltenek a helyi tár elsőszintű kitöltési területébe (9.7. táblázat). Ha ezt követően szelektorcsatorna-megszakítás érkezik be, az összes, a multiplexcsatorna működéséhez lényeges adatot a helyi tár másodikszintű tartományába töltik ki. Ezután kiszolgálják a szelektorcsatornát. Amikor valamennyi SCS művelet befejeződött, visszaállítják az MCS-hez tartozó adatokat és ellátják a multiplexcsatornát. Amint befejeződött az MCS kiszolgálása, visszaállítják a központi feldolgozóegység adatait és a CPU folytatja a feldolgozást.

A B típusú szelektorcsatornának saját áramköre van és így nincs időosztásos üzemben a központi feldolgozóegységgel; ezáltal egyidejűleg tud feldolgozást végezni a CPU-val. Nem használ helyi tárat pufferként, ezért elmarad a helyi tárhoz történő hozzáférés. Azonnal átviszi az adatokat a főtárhoz, amint két byte felhalmozódott. Ha két, egyidejűleg működő szelektorcsatorna van, egy-egy SCS maximális adatátviteli sebessége 300 000 byte/s. Ha csak egy SCS van, ennek a szelektorcsatornának maximális adatátviteli sebessége 400 000 byte/s. Ezeket az adatátviteli sebességeket is a 9.8. táblázatban foglaltuk össze. A szelektorcsatornákra vonatkozó további tárgyalást a B típusúra fogjuk korlátozni.

9.3.7. I/O interface (B/K illesztés)

A B/K vezérlőegységek kapcsolatát a csatornával az I/O interface (a B/K illesztés) teremti meg. Ezt az illesztést úgy szabványosítják, hogy tetszőleges B/K vezérlőegységet tetszőleges csatornához lehessen csatlakoztatni. Ezáltal gyakorlatilag az IBM összes B/K készüléke felhasználható a 40-es modellhez.

A szabványos illesztés (standard interface) — fizikailag — vezetékekből létrejövő adatút. Egy időegység alatt egy byte-nyi információt tud átvinni a csatorna és a B/K vezérlőegység között. Az illesztéshez csatolt valamennyi be-, ill. kimeneti készüléknél azonosak a villamos specifikációk (jelszintek, vonalmeghajtók és terminátorok).

Bármelyik típusú be-, ill. kimeneti készülékkel az átvitel ugyanazt az illesztési nyelvet használja fel. A B/K vezérlőegység ezt a nyelvet a hozzájuk csatlakozó be-, ill. kimeneti készülékek számára érthető aktuális vezérlőjelekké interpretálja (ill. fordítva), míg a csatornák az illesztési nyelvet fordítják le az adott központi feldolgozóegység vezérlőjelévé (ill. fordítva).

9.3.8.
Be-, ill. kimeneti vezérlőegységek és készülékek

A B/K vezérlőegységek működtetik a be-, ill. kimeneti készülékeket és fordítják a készüléknyelvet csatornanyelvre, és viszont. A 40-es modellben három általánosan alkalmazott B/K vezérlőegység van:

— integrált vezérlőegység;
— mágnesszalag-vezérlőegység és
— mágneslemez- (disk) vezérlőegység.

Az integrált vezérlőegység a lyukkártyaolvasót, lyukkártyalyukasztót és szimultán működésekhez a nyomtatót vezérli. A belső adatkódot kártyakódra fordítja le és viszont. Adatpuffereket szolgáltat egy teljes kártya- vagy nyomtatósorhoz. A mágnesszalag-vezérlőegység maximálisan nyolc mágnesszalag-működtetést vezérel. A mágneslemez-vezérlőegység maximálisan nyolc lemez működtetését vezérli. A soros bitekben megadott adatokat byte-okban összefogott adatokká fordítja és viszont.

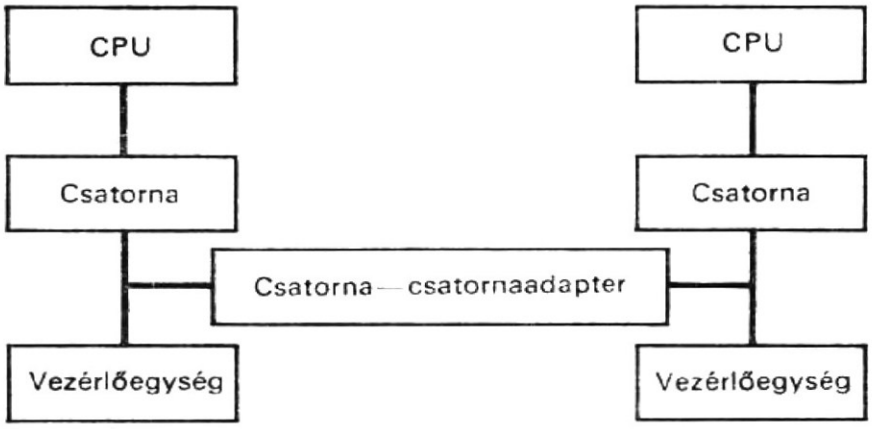
A be- ill. kimeneti készülék olyan eszköz, amely a lyukkártya vagy mágnesszalag-tárolóközeget továbbítja, továbbá vagy olvasási műveletet végez a közegről, vagy írási műveletet a közegre. Az általánosan használatos be-, ill. kimeneti készülékekre példa lehet a konzolírógép, a lyukkártyaolvasó—lyukasztó, a mágnesszalagtár, a mágnesdobtár, valamint a mágneslemeztár, az utóbbi akár állandó, akár cserélhető lemezköteggel.

A B/K vezérlőegység nem kell, hogy a B/K készüléktől független, külön álló egység legyen. Az IBM 1443-es nyomtató vezérlőegysége pl. a nyomtató része, a konzolírógép vezérlőegysége a konzolhoz tartozik és a mágnesszalag vezérlőegysége egy vagy több mágnesszalaghajtást tartalmazhat.

9.3.9.
Csatorna—csatornaadapter

Amint a 9.7. ábrán látható, az IBM csatorna—csatornaadaptere két csatornának, rendszerint két különböző számítógéprendszernek a B/K illesztéseit úgy kapcsolja össze, hogy mindegyik csatorna vezérlőegységnek tűnjön a másik számára. Minthogy az IBM System/360 a számítógépcsalád B/K illesztése szabványosított a csatorna—csatornaadapter arra is alkalmas, hogy a család egyik modelljét összekapcsolja egy másik modelljével.

Az adapter burst-üzemmódban működik. Egy byte-os adatot teljes szélességében visz át, a két csatorna által meghatározott sebességgel. Ha a két csatorna egyetlen számítógéprendszerben van, az adapter egy adatblokknak a főtár egyik helyéről a másikra való mozgatását teszi lehetővé. Ha két számítógéprendszerről van szó, az adapter a két rendszer két CPU-ját kapcsolja össze és így multiprocesszor rendszert képez.



9.7. ábra
Csatornák közötti kapcsolat

9.4. Feldolgozóegység

A feldolgozóegység az a hely, ahol az utasításokat végrehajtják. Az egyes utasítások leírása helyett ebben a szakaszban az utasításkészletet, az általános rendeltetésű regisztereket, a feldolgozási üzemmódokat, a feldolgozási műveleteket és a kódátalakításokat tárgyaljuk.

9.4.1. Utasításkészlet

A központi egység utasításkészletét a 9.9...9.12. táblázatokban találjuk meg. Ezeket az utasításokat négy típusra oszthatjuk:

— adatkezelő utasítások;
— elágazási utasítások;
— B/K utasítások;
— rendszervezérlő utasítások.

Az adatkezelő utasítások aritmetikai és logikai műveleteket hajtanak végre. A fixpontos, lebegőpontos, decimális aritmetikai utasítások és logikai utasítások tartoznak ide. Az elágazási utasításokkal változtatások hozhatók létre a program utasításainak sorrendjében. Ide a feltételes elágazási utasítások, az elágazó és összekapcsoló utasítások, a leszámlálásra elágazó utasítások és a végrehajtási utasítások tartoznak.

Négy *B/K utasítás* van:

— START I/O (B/K start);
— HALT I/O (B/K állj);
— TEST I/O (B/K teszt);
— TEST CHANNEL (csatornateszt).

A rendszervezérlő utasítások a teljes rendszerállapotot (státuszt) vezérlik. Ezek a következők: a PSW-PSSZ (programstátuszszó) betöltése, a programmaszk beállítása, a rendszermaszk beállítása, felügyelőprogram-hívás, a tárvédelmi kulcs beállítása, a tárkulcs inzertálása, közvetlen írás, közvetlen olvasás és diagnózis.

9.9. táblázat

Műveleti kód, RR formátum

	Elágazás és állapot-átkapcsolás	Fixpontos egész szavas és logikai utasítások	Lebegőpontos hosszúszavas utasítások	Lebegőpontos rövidszavas utasítások
xxxx	0000xxxx	0001xxxx	0010xxxx	0011xxxx
0000		Pozitív betöltése	Pozitív betöltése	Pozitív betöltése
0001		Negatív betöltése	Negatív betöltése	Negatív betöltése
0010		Betöltés és teszt	Betöltés és teszt	Betöltés és teszt
0011		Komplemens betöltése	Komplemens betöltése	Komplemens betöltése
0100	Program-maszk beállítása	ÉS	Felezés	Felezés
0101	Elágazás és összekapcsolás	Logikai összehasonlítás		
0110	Elágazás számlálás	VAGY		
0111	Elágazás/feltétel	Kizáró VAGY		
1000	Kulcs beállítása	Betöltés	Betöltés	Betöltés
1001	Kulcs inzertálása	Összehasonlítás	Összehasonlítás	Összehasonlítás
1010	Felügyelőprogram hívása	Összeadás	N összeadás	N összeadás
1011		Kivonás	N kivonás	N kivonás
1100		Szorzás	Szorzás	Szorzás
1101		Osztás	Osztás	Osztás
1110		Logikai összeadás	U összeadás	U összeadás
1111		Logikai kivonás	U kivonás	U kivonás

Megjegyzés: N=normalizált. U=nemnormalizált.

9.4.2. Általános rendeltetésű regiszterek

Az ideiglenes címekhez vagy adatokhoz tartozó általános rendeltetésű regiszterek alkalmazása megnövelheti a feldolgozási sebességet, mivel csökkenti a főtárhoz való hozzáférések számát. A feldolgozóegységnek 16, egyenként 32-bites általános rendeltetésű regiszterből álló készlete van, amelyeket *általános regisztereknek* neveznek és egy további opcionális készlettel rendelkezik, amely négy, 64-bites regiszterből áll. Ez utóbbi neve *lebegőpontos regiszter*. A lebegőpontos regiszterekben tárolják átmenetileg a lebegőpontos számokat a többszörös lebegőpontos aritmetikai műveletekhez. Az általános regisztereket indexregiszterekként, áthelyező (relokációs) regiszterként és akkumulátorként lehet használni a fixpontos aritmetikai vagy a logikai műveleteknél. Ezeket a regisztereket egy 4-bites címmel lehet megcímezni. Ha a regisztereket a program feldolgozása során más célra használják, tartalmukat a főtárban lehet tárolni, és ezekbe a regiszterekbe valamilyen más értéket lehet betölteni. Egyetlen utasítás áll rendelkezésre a regiszterkészlet betöltésére vagy tárolására. Amint az előzőkben említettük, az utasításban a 4-bites R mező címezi az általános

9.10. táblázat

Műveleti kód, RX formátum

	Fixpontos félszavas és elágazási utasítások	Fixpontos félszavas és logikai utasítások	Lebegőpontos hosszúszavas utasítások	Lebegőpontos rövidszavas utasítások
xxxx	0100xxxx	0101xxxx	0110xxxx	0110xxxx
0000	Tárolás	Tárolás	Tárolás	Tárolás
0001	Címbetöltés			
0010	Karaktertárolás			
0011	Karakterinzertálás			
0100	Végrehajtás	ÉS		
0101	Elágazás és összekapcsolás	Logikai összehasonlítás		
0110	Elágazás számlálás	VAGY		
0111	Elágazás/feltétel	Kizáró VAGY		
1000	Betöltés	Betöltés	Betöltés	Betöltés
1001	Összehasonlítás	Összehasonlítás	Összehasonlítás	Összehasonlítás
1010	Összeadás	Összeadás	N összeadás	N összeadás
1011	Kivonás	Kivonás	N kivonás	N kivonás
1100	Szorzás	Szorzás	Szorzás	Szorzás
1101		Osztás	Osztás	Osztás
1110	Decimálisra konvertálás	Logikai összeadás	U összeadás	U összeadás
1111	Binárisra konvertálás	Logikai kivonás	U kivonás	U kivonás

Megjegyzés: N=normalizált. U=nemnormalizált.

regisztereket. Ez a mező címezheti a lebegőpontos regisztereket is; mind ezt a műveleti kód specifikálja.

A 30-as modellben az általános és a lebegőpontos regiszterek a főtár részei. A 40-es és 50-es modellben ezek ugyancsak mágnesmagos, de gyorsabb tárolók. A 65-ös, 67-es és 75-ös modellben félvezetős regisztereket alkalmaznak, a még gyorsabb működés elérésére.

9.4.3. Feldolgozási üzemmódok

Három utasításvégrehajtási (feldolgozási) üzemmód van:

— regiszter—regiszter közötti;
— tár—regiszter közötti;
— tár—tár közötti.

A 40-es modellben a tár—tár közötti üzemmódban egy időegység alatt két byte-nyi operandust vesznek ki a főtárból és egyszerre egy byte-on végeznek műveletet. Az eredményt a főtárba kétbyte-onként írják be egy időegység alatt. Az operandusnak változó a hosszúsága: 1-től 16 byte lehet decimális szám esetén, és 1-től 256 byte karakterlánc (string) alkalmazásakor.

A tár—regiszter közötti üzemmódban a két operandus egyike a főtárban van, míg a másik az egyik regiszterben; ezek bináris operandusok. Ezeket az operandusokat az aritmetikai és logikai egység bemenetére viszik, műveletvégzés céljából és az ered-

9.11. táblázat

Műveleti kód, RS és SL formátum

	Elágazás, állapot-átkapcsolás és léptetés	Fixpontos logikai és be-, ill. kiviteli utasítások		
xxxx	1000xxxx	1001xxxx	1010xxxx	1011xxxx
0000	Rendszermaszk beállítása	Többszörös tárolás		
0001		Teszt maszk alatt		
0010	PSSZ (PSW) betöltése	Áthelyezés		
0011	Diagnózis			
0100	Direkt írás	ÉS		
0101	Direkt olvasás	Logikai összehasonlítás		
0110	Elágazás/nagyobb	VAGY		
0111	Elágazás/kisebb-egyenlő	Kizáró VAGY		
1000	SL léptetés jobbra	Többszörös betöltés		
1001	SL léptetés balra			
1010	S léptetés jobbra			
1011	S léptetés balra			
1100	DL léptetés jobbra	B/K start		
1101	DL léptetés balra	B/K teszt		
1110	D léptetés jobbra	B/K állj		
1111	D léptetés balra	Csatorna teszt		

Megjegyzés: N=normalizált. S=egyetlen. SL=egyetlen logikai. U=nemnormalizált. D=kettős. DL=kettős logikai.

9.12. táblázat

Műveleti kód, SS formátum

	Logikai utasítások			Decimális utasítások
xxxx	1100xxxx	1101xxxx	1110xxxx	1111xxxx
0000				
0001		Numerikus áthelyezés		Áthelyezés eltolással
0010		Áthelyezés		Tömörítés
0011		Zónaáthelyezés		Lazítás
0100		ÉS		
1010		Logikai összehasonlítás		
0110		VAGY		
0111		Kizáró VAGY		
1000				Törlés és összeadás
1001				Összehasonlítás
1010				Összeadás
1011				Kivonás
1100		Átalakítás		Szorzás
1101		Átalakítás és teszt		Osztás
1110		Szerkesztés		
1111		Szerkesztés és jelölés		

mény a regiszterben levő operandus helyére kerül. A regiszter—regiszter közötti üzemmódban ugyancsak mindkét operandus bináris és a regiszterekben van. Ebben az üzemmódban a főtár más rendszeregység kiszolgálására áll rendelkezésre. E két mód operandusai rögzített hosszúságúak. Félszavak, teljes szavak vagy kettősszavak lehetnek, de ez esetben ügyelni kell a címhatár egyeztetésre.

9.4.4. Feldolgozási műveletek

A feldolgozási műveleteknek négy osztálya van:
- fixpontos aritmetikai;
- lebegőpontos aritmetikai;
- decimális aritmetikai;
- logikai műveletek.

A fixpontos aritmetikai műveleteknél az operandus 32-bites, fixpontos bináris szám. A nagyobb sebesség vagy a jobb tárkihasználás érdekében félszóműveletet lehet a legtöbb műveletben specifikálni. Néhány szorzat és valamennyi osztandó 64 bit hosszú, páros—páratlan regiszterpárt használva. Minthogy a 32-bites szóhosszba elfér a 24-bites cím, a teljes fixpontos utasításkészletet használják a címkiszámításban.

Amint az előzőkben említettük, a lebegőpontos szám mind rövid, mind pedig hosszú formátumú lehet. A rövid formátum mintegy 7 decimális számjegy pontosságot ad, míg a hosszú formátum pontossága körülbelül 17 decimális számjegy. Mind a fixpontos, mind a lebegőpontos műveletek úgy kerülnek végrehajtásra, hogy az egyik operandus a regiszterből, a másik pedig egy másik regiszterből vagy főtárból olvasandó ki. Az eredmény, amelyet a regiszterben helyeznek el, rendszerint ugyanolyan hosszú, mint az operandus.

A decimális aritmetika összeadást, kivonást, szorzást, osztást és összehasonlítást végez. A decimális műveleteknél mind az operandusok, mind az eredmény a főtárban van. A decimális műveletekben a számok tömörített decimális formátumúak. A számjegyek tömörítése a tár hatékony kihasználását, az aritmetikai teljesítőképesség növelését segíti elő és javítja az adatátviteli sebességet.

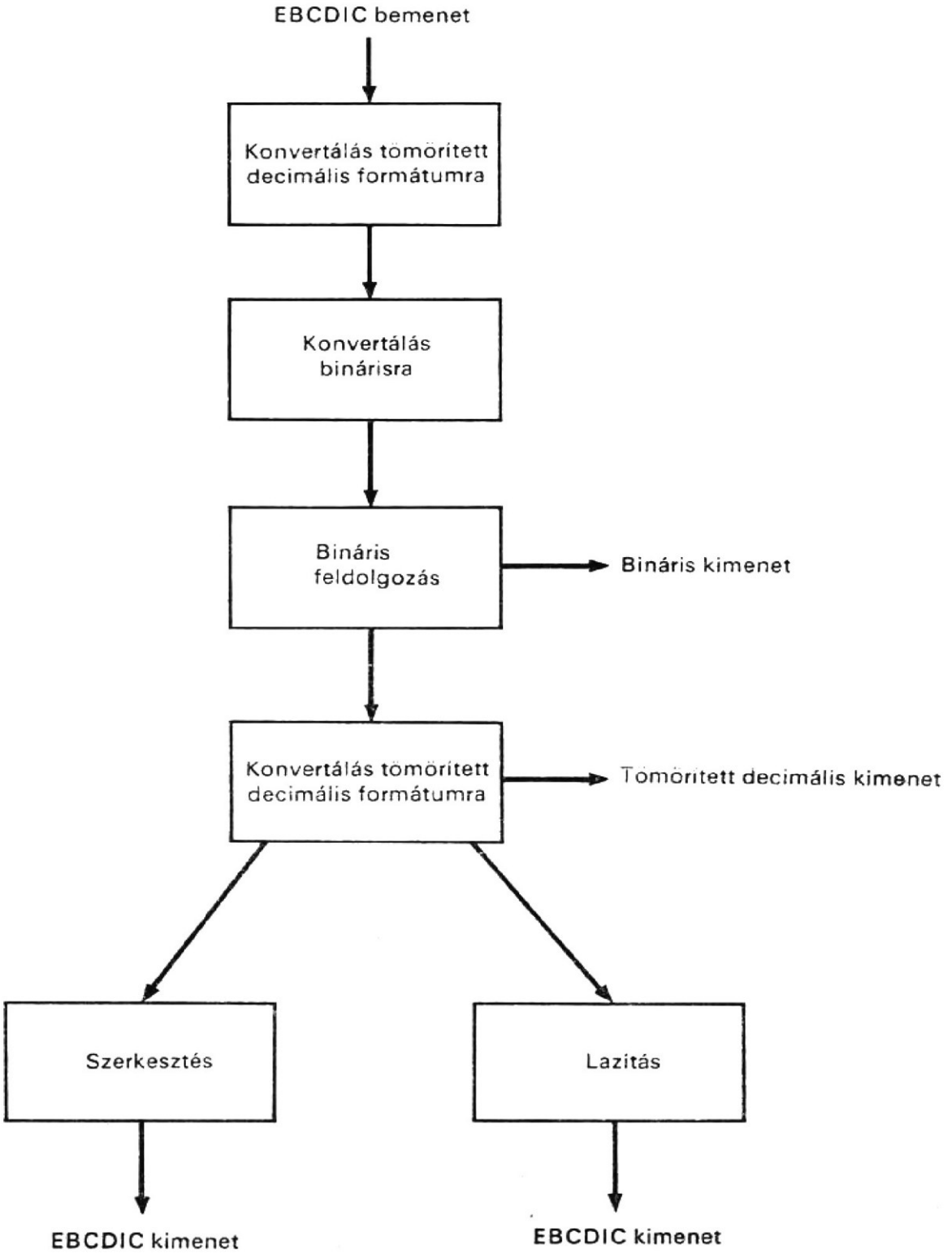
Összehasonlítási, fordítási, szerkesztési, bittesztelési és bitbeállító műveletek szolgálnak a rögzített vagy változó hosszúságú logikai operandusok feldolgozásához. A változó hosszúságú operandusokon végzett logikai műveleteket balról jobbra dolgozzák fel, időegységenként egy byte-ot. Két letapogató utasítás byte-ról byte-ra fordítást és tesztelést tesz lehetővé, táblázatok segítségével. A változó hosszúságú operandusokhoz a logikai műveletek egy fontos osztályát képezik azok, amelyek egybyte-os mező egyedi bitjeit tesztelik, beállítják, visszaállítják és komplementálják, az utasításban levő 8-bites maszk segítségével.

9.4.5. Kódátalakítás

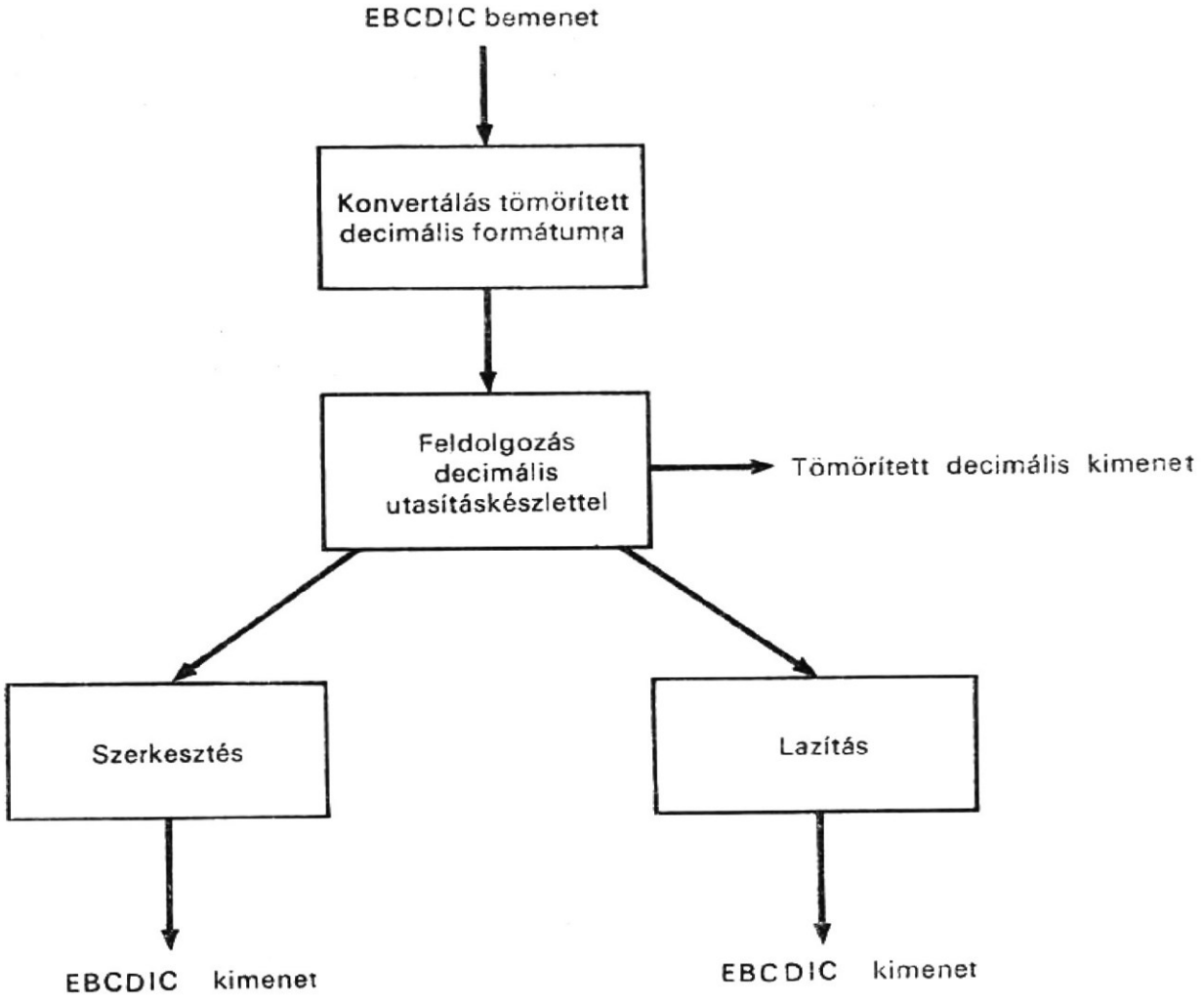
Amikor valamilyen karakterkészlettel rendelkező bemeneti készülékről EBCDIC kódban kódolt numerikus adatokat olvasnak be, ezeket a PACK utasítás segítségével, tömörített decimális formátumra alakítják (fordítják) át. Ha a numerikus adatok rögzített hosszúságúak, bináris aritmetikát használnak, míg változó hosszúságú adatoknál decimális aritmetikát. Ha bináris aritmetikát használtak, binárisra—konvertálási utasítás fordítja le az adatokat fixpontos formátumra és ezután fixpontos aritmetikai

műveletek dolgozzák fel őket. A feldolgozás után decimálisra—konvertálási utasítást, vagy lazítási (unpack) utasítást, vagy szerkesztési (edit) utasítást alkalmaznak. Ezek az utasítások fordítják le az adatokat EBCDIC kódba, valamilyen kimeneti készülék, pl. nyomtató vagy lyukasztó számára (9.8. ábra). Ha decimális aritmetikát használtak, a konvertálási utasítások elhagyhatók és az adatokat decimális aritmetikai utasítások dolgozzák fel (9.9. ábra).

Abban az esetben, ha a bemenet más kódban (max. nyolc szóhosszal) fordul elő, például távírókódban, a kódfordítást a tárban levő fordítási táblázat felhasználásával lehet végrehajtani. A fordítási utasítás maximálisan 256-karakteres string (lánc) konverzióját teszi lehetővé.



9.8. ábra
EBCDIC bemenetű, fixpontos aritmetikai feldolgozási sorozat



9.9. ábra
EBSDIC bemenetű, decimális utasításkészletet felhasználó feldolgozás

9.5. A központi vezérlőegység

A központi vezérlőegység hajtja végre egy program utasításainak sorrendbe állítását, egyik programról a másikra való vezérlésátadást és a megszakítások kezelését, ha ilyenek előfordulnak. E szakaszban bemutatjuk ezeknek a funkcióknak a megvalósítását.

9.5.1. Az utasítások sorrendje

A központi vezérlőegység a program utasításait rendszerint úgy állítja sorrendbe, hogy a főtárból kiolvassa az utasításszámláló által megcímzett következő utasítást. Az utasítás lehívása után az utasításszámlálót az utasítás byte-jainak számával megnövelik; a számláló most a következő utasítás címét tartalmazza. Ezt a normál sorrendet elágazási utasítással vagy megszakítással meg lehet változtatni. A megszakítással történő sorrendváltoztatásról a későbbiekben lesz szó.

Az elágaztatási utasításokkal történő sorrendváltozást a leggyakrabban azzal a feltételes elágazási utasítással végzik, amely a 2-bites Y2—Y3 feltételregiszter állapotát

felügyeli. Több aritmetikai, logikai és be-, ill. kimeneti művelet úgy jelzi az eredményt, hogy a feltételregiszter négy állapotának egyikét állítja be. Ebből következik, hogy a feltételes elágazás ki tudja választani az egyik állapotot elágazási kritériumként. A feltételkód pl. olyan állapotokat jelezhet, mint túlcsordulás, egyenlőség, nemnulla vagy csatornafoglaltság.

A címaritmetikai és leszámlálási műveletek eredményét feltételes elágazással lehet ellenőrizni, hurok vezérlése céljából. Két utasítás az elágazásleszámlálásra és az elágazásindexre egyutasításos végrehajtást biztosít a leggyakoribb aritmetikai és ellenőrzési műveletkombinációkhoz. A szubrutin-összekapcsolást az elágazás- és összekapcsolás-utasítás végzi el. A végrehajtás-utasítás a pillanatnyi utasítássorozatban nem szereplő, egyetlen utasítás végrehajtását engedélyezi.

9.5.2. CPU állapot

A CPU állapotát négy állapothalmaz jelöli ki:

— a stop- vagy működésállapot;
— a futás- vagy várakozásállapot;
— a maszkolt vagy megszakítható állapot;
— a felügyelőprogram (supervisor) vagy feladatállapot.

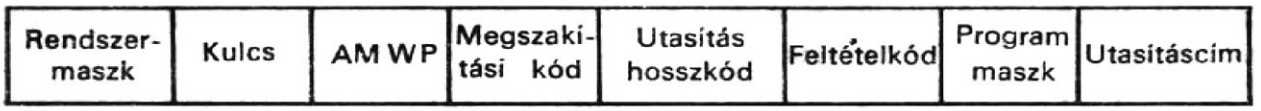
A *működésállapot* azt jelzi, hogy a központi feldolgozóegység képes utasítások végrehajtására, a megszakításokat elfogadja és az időadót felfrissítette (időmérőt vezérelte). Ha a CPU stop-állapotban van, az utasításokat nem hajtja végre, megszakításokat nem fogad el, és az óra nincs felfrissítve. Ezeket az állapotokat kézzel vezérlik.

A *futásállapot* azt jelzi, hogy az utasításokat a CPU normál módon éppen végrehajtja. Ha a CPU várakozásállapotban van, nem hajt végre utasítást, de az óra fel van frissítve és elfogadja a B/K- és külső megszakításokat, hacsak azok nincsenek maszkolva. *Várakozásállapotba* rendszerint azért lép a program, hogy megszakításra várjon, pl. B/K megszakításra vagy gépkezelői beavatkozásra.

Amikor a CPU *megszakítható állapotban* van, megszakításokat engedélyez. A legtöbb megszakítás maszkolható.

9.13. táblázat
Privilegizált utasítások

Csoport	Privilegizált utasítás
PSW művelet	PSW tárolása
	Rendszermaszk beállítása
B/K művelet	Start I/O, B/K start
	Stop I/O, B/K stop
	Test I/O, B/K teszt
	Test Channel, csatornateszt
Tárvédelem	Tárolókulcs inzertálása
	Tárolókulcs beállítása
Közvetlen vezérlés	Közvetlen olvasás
	Közvetlen írás
Diagnózis	Diagnosztizálás



0—7 Rendszermaszk
0 Multiplexcsatorna-maszk
1 1 szelektorcsatorna-maszk
2 2 szelektorcsatorna-maszk
3 3 szelektorcsatorna-maszk
4 4 szelektorcsatorna-maszk
5 5 szelektorcsatorna-maszk
6 6 szelektorcsatorna-maszk
7 Külső/időadó maszk

8—11 Védelmi kulcs
12 ASCII üzemmód (A)
13 Gépellenőrzés-maszk (M)
14 Várakozásállapot (W)
15 Problémaállapot (P)

16—31 Megszakításkód
32—33 Utasításhosszkód (ILC)
34—35 Feltételkód (CC)
36—39 Programmaszk
36 Fixpontos-túlcsordulás maszk
37 Decimális-túlcsordulás maszk
38 Kitevő-alulcsordulás maszk
39 Szignifikanciamaszk
40—63 Utasításcím

9.10. ábra
Programállapotszó (PSW) formátuma

Amikor a CPU *felügyelőállapotban* van, valamennyi utasítás érvényes és végrehajtható. Ha a CPU feladatállapotban van, az utasítások egy csoportja, amelyeket *privilegizált utasításoknak* neveznek, nem adható ki és ezen utasítások végrehajtása programellenőrzési megszakítást eredményez. A privilegizált utasításoknak öt csoportja van, amint az a 9.13. táblázatban látható.

9.5.3.
Programállapotszó (Program Status Word)

A programállapot a feladatállapot teljes rendszerbeli állapotára vonatkozik. Valamennyi, a programállapot jelzését és vezérlését szolgáló információt egy kettős szó, az ún. *programállapotszó* PSW(PSSZ) tárolja. A PSW(PSSZ) formátumát a 9.10. ábrán láthatjuk. A PSW alkalmazásának az az alapötlete, hogy hardware segítségével úgy rögzítjük a gép állapotát, hogy a központi vezérlőegység bármikor készen legyen a programátkapcsolás végrehajtására.

A PSW-ben nyolc mező van:
- rendszer maszk (0...7);
- védelmi kulcs (8...11);
- AWMP (12...15);
- megszakítási kód (16...31);
- utasításhosszkód (32...33);
- feltételkód (34...35);
- programmaszk (36...39);
- a következő utasítás címe (40...63).

Ezeknek a mezőknek értelmezését az alábbiakban írjuk le.

1. Rendszermaszk. A 0 bitben a 0 érték maszkolja a multiplexcsatornából jövő megszakításokat (azaz megakadályozza a megszakítások létrejöttét). Az 1...6 bitekben a 0 érték az 1...6 szelektorcsatornákból érkező megszakításokat maszkolja le. A 7. bitben levő 0 érték a külső vagy óra- (timer) megszakítást maszkolja le.

2. **Tárvédelmi kulcs.** 4-bites tárvédelmi kulcs, amely ahhoz a programhoz van hozzárendelve, amelyet a PSW képvisel.

3. **AMWP.** Az A, M, W és P regiszterek rendre a 12-től 15-ig terjedő biteket képviselik. Az A bitben az 1 érték azt jelzi, hogy ASCII—8 kódot alkalmaznak; egyébként pedig EBCDIC kódot jelöl. Az M bitben a 0 érték a gépellenőrzési (machine check) megszakítást maszkolja le. A W bitben az 1 érték azt jelzi, hogy a CPU várakozási álla-

9.14. táblázat

A PSW megszakítási kódjának értelmezése

A megszakítás forrása	Megszakítási kód (a PSW 16—31 bitjei)		Maszk (PSW bit)
	B/K megszakítás		
Multiplexcsatorna	00000000	aaaaaaaa	0
1. szelektorcsatorna	00000001	aaaaaaaa	1
2. szelektorcsatorna	00000010	aaaaaaaa	2
3. szelektorcsatorna	00000011	aaaaaaaa	3
4. szelektorcsatorna	00000100	aaaaaaaa	4
5. szelektorcsatorna	00000101	aaaaaaaa	5
6. szelektorcsatorna	00000110	aaaaaaaa	6
	Programmegszakítás		
Művelethiba	00000000	00000001	
Privilegizált művelet megsértése	00000000	00000010	
Végrehajtási hiba	00000000	00000011	
Védelemmegsértés	00000000	00000100	
Címzési hiba	00000000	00000101	
Specifikációs hiba	00000000	00000110	
Adathiba	00000000	00000111	
Fixpontos túlcsordulás	00000000	00001000	36
Fixpontos osztás hiba	00000000	00001001	
Decimális túlcsordulás	00000000	00001010	37
Decimális osztás hiba	00000000	00001011	
Kitevő túlcsordulás	00000000	00001100	
Kitevő alulcsordulás	00000000	00001101	38
Szignifikancia hiba	00000000	00001110	39
Lebegőpontos osztáshiba	00000000	00001111	
	Felügyelőprogram-hívás megszakítás		
Utasításbitek	00000000	rrrrrrrr	
	Külső megszakítás		
7 külső jel	00000000	xxxxxxx1	7
6 külső jel	00000000	xxxxxx1x	7
5 külső jel	00000000	xxxxx1xx	7
4 külső jel	00000000	xxxx1xxx	7
3 külső jel	00000000	xxx1xxxx	7
2 külső jel	00000000	xx1xxxxx	7
Megszakítási kulcs	00000000	x1xxxxxx	7
Óra	00000000	1xxxxxxx	7
	Gépellenőrzési megszakítás		
Gépi meghibásodás	cccccccc	cccccccc	13

Megjegyzés:
aaaaaaaa jelöli a megszakítást okozó B/K egység készülékének címét.
rrrrrrrr jelöli a felügyelőprogram-hívás (supervisor call — SVC) utasítás 8...15 bitjét.
cccccccc modellfüggő kód.

potban van; egyébként pedig futási állapotban. A P bitben az 1 érték jelentése az, hogy a CPU feladat állapotban van; egyébként pedig felügyelő- (supervisor) állapotban.

4. Megszakítási kód. A megszakítási kód a megszakítás forrását azonosítja. A kód jelentése az öt megszakítási csoport szerint változik. Ezeket az értelmezéseket, valamint a PSW maszkbitjeit, ha alkalmazhatók, a 9.14. táblázat mutatja.

5. Utasításhosszkód. Az utasításhosszkód a legutoljára végrehajtott utasítás hosszát azonosítja; ez az információ bizonyos megszakítások esetén azért szükséges, hogy meg lehessen határozni azt az utasítást, amelyet éppen interpretáltak, amikor a megszakítás történt.

6. Feltételkód. A feltételkódot bizonyos aritmetikai, összehasonlító, logikai és B/K utasítások állítják be. A kód jelentése különböző, az egyes utasításoktól függően. Ahhoz, hogy némi fogalmat adjunk a kódról, a 9.15. táblázatban bemutatjuk az egyes műveletek jelentését.

9.15. táblázat

A PSW feltételkódjának értelmezése

Utasítás	Feltételkód			
	00	01	10	11
Aritmetikai műveletek	0	<0	>0	túlcsordulás
Összehasonlítási műveletek	egyenlő	kisebb	nagyobb	—
Logikai műveletek*	0	$\neq 0$	—	—
B/K műveletek**	elérhető	csatornaállapot elérhető	foglalt vagy lezárt	nem működő

* Logikai ÉS, VAGY, kizáró VAGY műveletekhez tartozó utasítások.
** A feltételkód a megcímzett egységre vonatkozik.

7. Programmaszk. A programmaszk a program számára teszi lehetővé a fixpontos túlcsordulás-megszakítás (36. bit), a decimális-túlcsordulás-megszakítás (37. bit), a kitevő-alulcsordulás-megszakítás (38. bit) és a szignifikanciamegszakítás (39. bit) maszkolását.

8. Utasításcím. Ez a következő utasítás címe. Ha ezt az utasítást először helyezik el a PSW-ben, ez az első utasításcím. Minden egyes utasítás lehívásakor ezt a címet felfrissítik. Az elágazási címmel helyettesítik egy sikeres elágazási utasítás végrehajtása után.

Az öt megszakítási osztály mindegyikének két helye van, két programállapotszónak, a *régi PSW-nek* és az *új PSW-nek* a tárolására. Ezek állandó helyek a főtárban, a 24, 32, 40, 48, 56, 88, 96, 104, 112 és 120 címen (9.16. táblázat).

9.5.4. Megszakítás

A megszakítási rendszer olyan hardware-rendszer, amely megszakítás felléptekor lehetővé teszi egyik programról egy másikra történő áttérést. Ebben a fejezetben a megszakítások típusait, maszkolhatóságát és prioritását, valamint a programátkapcsolás mechanizmusát írjuk le.

9.16. táblázat

Állandó tárhelyek kijelölése

Cím	Byte-hossz	Rendeltetés
0	8	Kezdeti program betöltő PSW
8	8	Kezdeti program betöltő CCW 1
16	8	Kezdeti program betöltő CCW 2
24	8	Külső régi PSW
32	8	Felügyelőprogram-hívás régi PSW
40	8	Program régi PSW
48	8	Gép ellenőrzés régi PSW
56	8	Be/kivitel régi PSW
64	8	Csatornaállapotszó
72	4	Csatornacímszó
76	4	Nincs felhasználva
80	4	Óra
84	4	Nincs felhasználva
88	8	Külső új PSW
96	8	Supervisor-hívás új PSW
104	8	Program új PSW
112	8	Gépellenőrzés új PSW
120	8	Be/kivitel új PSW
128		Diagnosztikai terület

* A diagnosztikai terület mérete struktúrafüggő.

A megszakításoknak öt típusa van:

a) géphiba;
b) külső;
c) B/K;
d) programhiba;
e) supervisor (felügyelőprogram) -hívás.

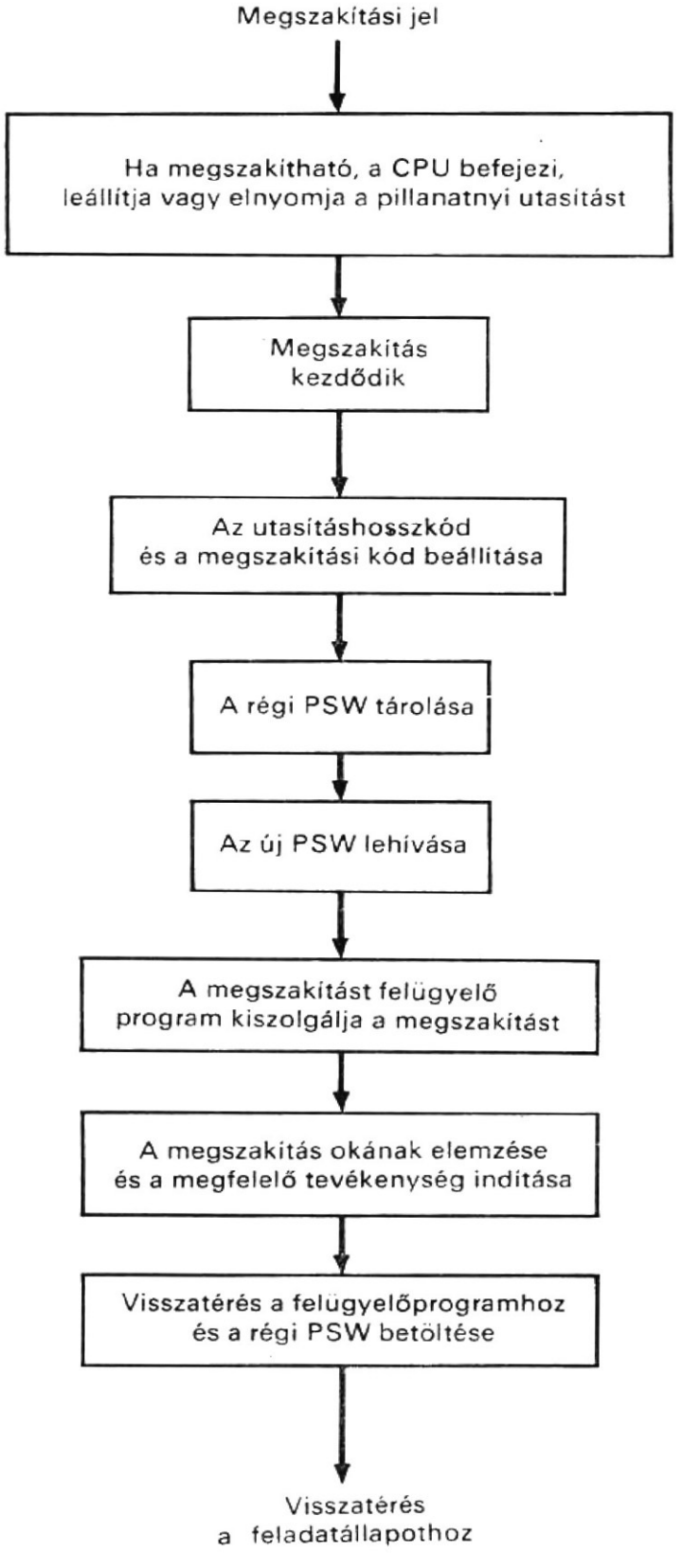
Géphiba (vagy gépellenőrzés) *megszakítás* akkor lép fel, ha hardware-meghibásodás fordul elő. Ez a megszakítás olyan rutint indít meg, amely meghatározza a hibát és kiírja a központi feldolgozóegység állapotát (automatikusan feljegyzi a hardware állapotát a tárban) a 128 különböző tárhelyen és a PSW-ben (9.16. táblázat). A *külső megszakítás* lehetőséget ad a CPU-nak, hogy a külső jelekre, pl. a rendszervezérlő kezelőlapon levő megszakítási kulcsról és a hardware-órától jövő jelekre válaszoljon. A *B/K megszakítás* segítségével a CPU válaszolni tud a csatornában, vagy a be-, ill. kimeneti egységekben előforduló különböző feltételekre. A *programhiba* (vagy programellenőrzés) *megszakítás* akkor jön létre, amikor meg nem engedhető állapot fordul elő a feladatállapotban, mint pl. a túlcsordulás, helytelen osztás, elveszett előjel, tilos műveleti kód, privilegizált utasítás vagy a tárvédelem megsértése. A *felügyelőprogram (supervisor)-hívás megszakítás* akkor jön létre, ha a feladatállapotban felügyelő program-hívás utasítást hajt végre; ennek eredménye átkapcsolás a problémaállapotról felügyelőprogram-állapotra.

A megszakítás első típusa a PSW 13 bitjével maszkolható. A második típus a PSW bitjével maszkolható. A harmadik típus a PSW rendszermaszkjával (0...6 bit) maszkolható. A negyedik típusnak van egy részhalmaza, amely a PSW programmaszkjával (36...39 bit) maszkolható. Az ötödik típusnak nincs szüksége a maszkolhatóságra és így nem is maszkolható.

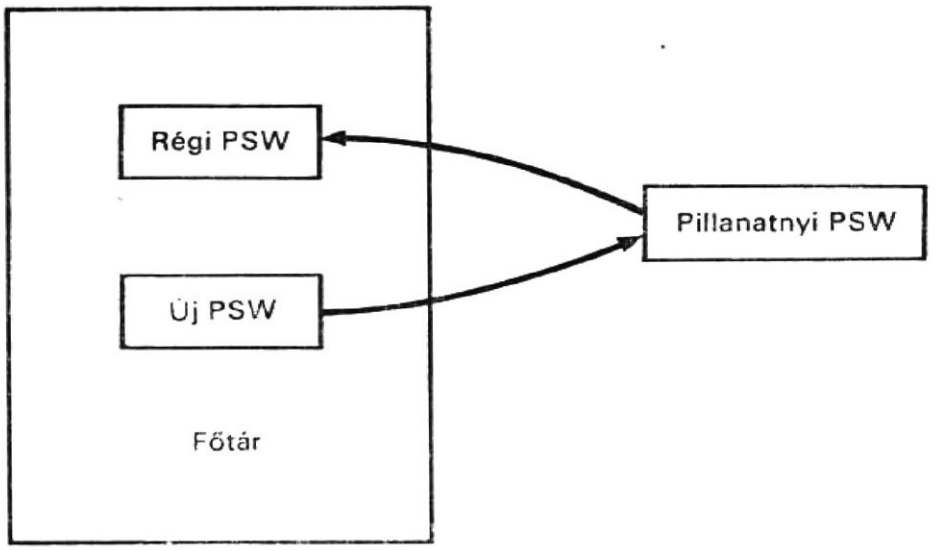
Valamely utasítás végrehajtása során, ha egynél több, különböző típusú megszakítás jön létre egyidőben, csak egyetlen megszakítás fogadható el egyszerre. A szimultán megszakításokat a következő prioritási sorrendben fogadja el a megszakítási rendszer:

1. Gépellenőrzési megszakítás.
2. Programellenőrzési vagy felügyelőprogram-hívási megszakítás.
3. Külső megszakítás.
4. B/K megszakítás.

Jegyezzük meg, hogy a programellenőrzési és a felügyelőprogram-hívási megszakítások kölcsönösen kizárják egymást, mivel nem jöhetnek egyszerre létre.



9.11. ábra
Megszakítás folyamata



9.12. ábra
Programállapotszó (PSW) átkapcsolása a megszakítás alatt

A megszakítás létrejötte utáni CPU tevékenységet a 9.11. folyamatábra szemlélteti. Amint az ábra mutatja, amikor megszakítás lép fel, a CPU meghatározza, hogy a jelenlegi állapot megszakítható-e. Ha igen, a pillanatnyi utasítást befejezi, leállítja vagy elnyomja és megkezdődik a megszakítási tevékenység. A CPU beállítja az utasításhosszkódot és a megszakítási kódot a pillanatnyi PSW-ben (amely a pillanatnyi feladatállapotban futó programot vezérli). Ezután a pillanatnyi PSW-t a régi PSW helyére tárolja és lehívja a PSW-t az új PSW helyéről, és így ebből lesz a pillanatnyi PSW. Pl. a régi és az új PSW a programban a főtár 40., ill. 104. helyén van, amint a 9.16. táblázatból látható. A programállapotszavaknak ezt a cseréjét *PSW átkapcsolásnak* nevezik, ennek megvalósítása a 9.12. ábrán látható. Az új PSW meghatározza a megszakítási rutint az adott megszakítási osztályhoz. Ezután a megszakítási rutin, amely állandóan a főtárban van, folytatja a megszakítás okának elemzését a régi PSW helyén levő programállapotszó alapján és elvégzi a megfelelő tevékenységet. Miután a megszakítást kiszolgálta, a megszakítási rutin utolsó utasítása egy PSW-betöltés utasítás, amely vissza behívja a régi PSW helyén levő programállapotszót és pillanatnyi PSW-vá teszi. Ily módon az eredeti feladatállapothoz tartozó program újból indulhat. Multiprogramozás esetén a PSW-betöltés utasítás egy másik PSW-t is hívhat.

A megszakítási rutin végrehajtása alatt felléphet egy újabb megszakítás. Ez a második megszakítás az első megszakítási rutinhoz tartozó pillanatnyi PSW tárolását idézheti elő. Tehát, elvész az eredeti feladatprogram régi PSW-je és nem lenne többé semmilyen módon visszatérés a feladatállapothoz tartozó programhoz. Ezért meg kell előzni egy megszakítási rutin végrehajtása során annak megszakítását egy ugyanolyan osztályú másik megszakítás által. Ezt a megelőzést az új PSW-ben levő megfelelő bitek *maszkolásával* érik el. A maszkolás segítségével a megszakítás felfüggeszthető későbbi végrehajtásig. Mint említettük, a PSW szolgáltatja a rendszermaszkot (0...7 bit), a gépellenőrzési maszkot (13. bit) és a programmaszkot (36...39 bit). Amikor valamilyen adott megszakítási típus maszkbitjeinek 0 értéket adnak, ezek a megszakítások nem fordulhatnak elő, azaz maszkolva vannak. Amikor a maszkbitek értéke 1, a CPU megszakítható a megfelelő megszakítási típusok számára.

9.5.5. Mikroprogram

Amint az előzőekben említettük, a gépen belül van egy csak olvasható tár (ROM). Minden ROM szó egy mikroutasítás és a mikroutasítások összessége alkotja a mikroprogramot. A mikroprogram és a hozzá tartozó áramkörök képezik a 40-es modell központi vezérlőegységének magját. A mikroutasításra vezérlőszóként is lehet hivatkozni. Minden vezérlőszónak 56 bitje van és 20 mezőre van osztva.

A mikroutasítások sorozata mikroszubrutint alkot. A mikroprogramban egy sor mikroszubrutin van. Megemlítünk itt néhány mikroszubrutint. A lehívási mikroszubrutin kiolvassa a végrehajtandó utasítást az utasításszámláló (amely a pillanatnyi PSSZ utasításcímmezője) által meghatározott főtárhelyről, dekódolja a műveleti kódot, elágazik a megfelelő mikroszubrutinhoz és ezután előállítja az utasításszámláló új értékét. A „mozgatási" (dump) vagy „visszamozgatási" (undump) mikroszubrutin a központi feldolgozóegység adatáramát a helyi tár kitöltési területén tárolja, ill. fordítva. A kezdeti-program betöltő mikroszubrutin 24 byte-nyi adatot tölt be arról a be-, ill. kimeneti készülékről, amelyet a konzolon levő betöltőkapcsolóval választottak ki. Ezt a 24 byte-ot a főtár 0-tól 23. helyére tölti be és egy parancsláncot (string-et) kezdeményez a 8. helyen levő csatornavezérlő-szóhoz. A szelektorcsatorna pufferkiszolgáló mikroszubrutinja „főtárciklust lop", és ezzel az adatokat a csatornából a főtárba viszi vagy megfordítva.

9.5.6. Megszakítás-felügyelőprogram

Amint a 9.11. ábrán látható, miután a PSW csere megtörtént, a megszakítás-felügyelőprogram a megszakítás típusának megfelelően kezeli a megszakítást. Ha az egy B/K megszakítás, a megszakítás-felügyelőprogram a B/K felügyelőprogramnak adja át, amely a B/K megszakításokat kezelő rutint tartalmazza. Ha a B/K megszakítás valamilyen B/K művelet előírt befejezését jelzi, a B/K felügyelőprogram beindít egy felfüggesztett B/K műveletet és azután visszaadja a vezérlést a problémaállapothoz tartozó programnak. Ha a B/K megszakításhibát jelez, valamennyi megszakítást lemaszkolnak és a vezérlést az adott B/K készülékhez tartozó hibajavító rutin kapja meg. Abban az esetben, ha nem lehetséges a javítás, a hibát nem veszik tekintetbe, a rekordot megkerülik vagy a vezérlést átadják egy felhasználói rutinnak, esetleg leállítják a munkát.

Ha gépellenőrzési megszakítás következik be, a megszakítást felügyelő program a CPU-t várakozási állapotba helyezi. Ezzel lehetővé teszi az operátor számára, hogy egy egyedi programot töltsön be és hajtson végre, amely megőrzi a kivizsgálandó terület tartalmát és egyéb rendelkezésre álló információt; nyomtatott kimenet vagy mágneslemezen vagy -dobon levő rekordok alakjában.

Minthogy a felügyelőprogram-hívás megszakítást a problémaállapothoz tartozó programból jövő kéréssel, a felügyelőprogram-hívás (supervisor-call, SVC) utasítással idézik elő, a megszakítás-felügyelőprogram átadja a vezérlést az SVC megszakítási rutinnak. Ez a rutin megvizsgálja az SVC utasítással együtt megadott megszakításkódot és ezután átadja a vezérlést a megfelelő rutinnak, a kérés kielégítése végett. A megfelelő rutinokra példa lehet egy programnak a könyvtárból a tárba való betöltése, végrehajtás végett; a gépkezelő és a feladatállapothoz tartozó program közötti kommunikációt biztosító rutin; valamint a program leállítására és a következő munka futtatásának előkészítésére szolgáló rutin.

Ha a külső megszakítást az időintervallumóra okozza, a vezérlés egy, a felhasználó által megadott időadó rutinhoz kerül; egyébként pedig a megszakítást nem veszik figyelembe. Ha a külső megszakítást a konzolon levő megszakítási kulccsal okozzák, a vezérlés átadódik az operátor kommunikációs rutinhoz, amely az operátortól a felügyelőprogramhoz érkező üzeneteket, vagy a felügyelőprogramtól az operátorhoz címzett üzeneteket látja el.

Programellenőrzési megszakítás esetén a megszakítás felügyelőprogramja átadja a vezérlést egy, a felhasználó által meghatározott „programhiba-elrendező rutin" címére; egyébként pedig a problémaprogram lezárul.

9.6. A felügyelőprogram és egyéb vezérlési módok

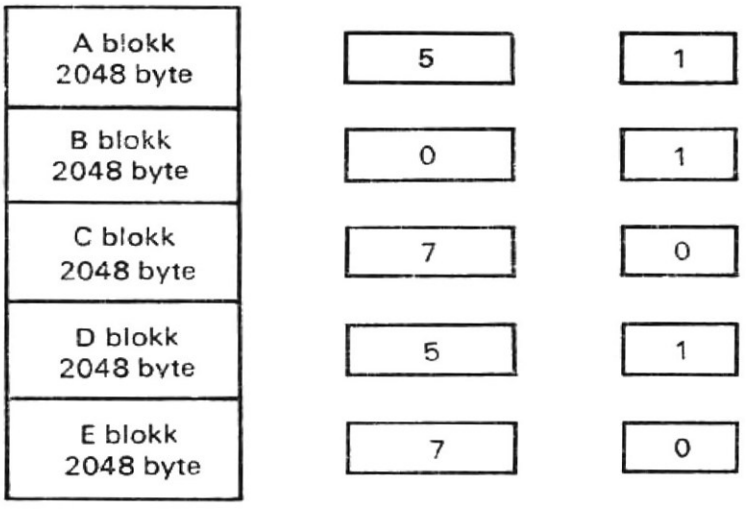
Amint növekszik a digitális számítógépek sebessége, ugyanolyan mértékben nő a teljesen automatikus rendszer iránti igény, hogy teljes mértékben ki lehessen használni a számítógép lehetőségeit. Parancsoló szükséglet tehát, hogy a számítógéprendszert felügyelőprogram vezérelje és a manuális vezérlést a minimumra szorítsuk. Ebben a fejezetben néhány hardware-lehetőséget mutatunk be, amelyek a felügyelőprogrammal végzett vezérléshez nyújtanak segítséget. Ezek a hardware-lehetőségek a következők: tárvédelem, a felügyelőprogram túlélésének biztosítására; időintervallum-adó, amely időszakosan visszaadja a vezérlést a felügyelőprogramnak; valamint a várakozási állapot kezelése. Ez utóbbi a felügyelőprogram rendelkezésére áll, míg a stop- vagy halt-utasítás használata a problémaállapotot használó a programozó eszköze. A továbbiakban röviden leírjuk a közvetlen vezérlést és a kezdeti-program betöltést.

9.6.1. Tárvédelem

A tárvédelem céljából a főtárat egyenként 2048 byte-os blokkokra osztják. Minden blokknak van egy 4-bites tárvédelmi kulcsregisztere. Ezt a négy bitet *tárkulcsnak* nevezik és a blokk zárjának tekinthetők. Az egyes blokkok tárkulcsait a felügyelőprogram „illeszti be".

A zár felnyitásához egy *védelmi kulcsra* van szükség. Minden programhoz hozzárendelnek egy védelmi kulcsot és ezt a programállapotszó 8...11. bitjében tárolják. A csatornaprogram védelmi kulcsát a csatornacímszó (CCW) 0...3. bitjében tárolják Amikor az adatokat valamelyik tárblokkban elhelyezik, összehasonlítják a tárkulcsot és a védelmi kulcsot. Ha ezek egyformák, vagy bármelyik 0, „összeillenek", és a blokk *tárolását* engedélyezik. Ha illesztetlenség fordul elő annak következtében, hogy egy utasítást védett területen akartak végrehajtani, az utasítás végrehajtását elnyomják vagy lezárják, és megszakítás generálódik. Ha az illesztetlenség be-, ill. kimeneti művelet végrehajtásának eredményeképpen jön létre, az adatátvitelt lezárják. Mindegyik esetben védett lesz a főtár tartalma (azaz változatlan marad).

Pl. legyen a 9.13. ábrán látható A, B, C, D és E blokk tárkulcsa rendre 5, 0, 7, 5 és 7. A felügyelőprogram PSW-jében a védelmi kulcs 0; ezen öt blokk bármelyikében lehet



9.13. ábra
Tárvédelem

szavakat tárolni. Ha a feladatállapothoz tartozó program PSW-jében a védelmi kulcs 5, akkor az A és D blokkokban megengedett a szavak tárolása, a B, C és E blokkokban viszont a tárolási kísérlet programellenőrzési megszakítást okoz. Ha a csatornacímszóban (CAW, ill. CCSZ) a védelmi kulcs 7, akkor a C és E blokkokba történő adatátvitel megengedett, de a többi blokkba történő adatátvitel B/K megszakítást eredményez.

A fentiekben leírt tárvédelem a tárolásra szolgál. Az IBM System/360 számítógépeinek néhány modelljében ezen kívül még lehívási védelem is van. Ehhez egy egybites kulcsregiszterrel látják el az egyes 2048-byte-os blokkokat. Amikor a kulcsregiszter 0 értéket tartalmaz, a blokk védve van az ellen, hogy információt írjanak belé. Ha a kulcsregiszter tartalma 1, a blokk védve van információlehívás (olvasás), valamint új információ tárolása (írás) ellen. Például a 9.13. ábrán látható A, B, C, D és E blokkok lehívási kulcsai rendre 1, 1, 0, 1 és 0. Az A, B és D blokknak egyaránt tárolási és lehívási védelme van, míg a C és E blokk csak tárolási védelemmel rendelkezik.

A fenti tárolási védelem lehetőséget ad a felügyelőprogramnak, hogy változtatásokat végezzen a főtár bármelyik blokkjában, míg a feladatállapothoz tartozó programnak csak a számára kijelölt blokkokban engedélyez változtatásokat.

9.6.2. Időintervallum-adó (timer)

A főtár 80-as helyén a teljes szó az időintervallum-adó által beállított időt tartalmazza. A maximális intervallum 15,5 óra. Ezt az időt a felügyelőprogram állítja be és másodpercenként 50- (vagy 60)-szor kerül beállításra. Amikor az időintervallum-adóban levő idő pozitívból átbillen negatívba, külső megszakítás jön létre, és ez jelzést ad a központi feldolgozóegységnek. Ezt az időintervallum-adót a munkák könyvelésére lehet használni, segítségével mérhetők az egyes munkákra fordított időtartamok, a túlfutott munkák leállíthatók, felhasználható a napi idő feljegyzésére, továbbá, a távadatfeldolgozó hálózatban levő adatállomások lekérdezését végezheti (polling) bizonyos időszakonként.

9.6.3.
Várakozási állapot

Amikor a PSW, ill. PSSZ 14-es bitje 1, a központi feldolgozóegység várakozási állapotban van. Nem végez tárhivatkozást és nem hajt végre utasítást, de reagál minden lehetséges megszakításra; azonnal válaszol, ha hívják. A várakozási állapot tehát lehetővé teszi a CPU-nak, hogy a rendszer vezérlése alatt maradjon akkor is, ha nincs végrehajtandó feladatállapotban levő program.

9.6.4.
Közvetlen vezérlés

A közvetlen olvasás (READ DIRECT) és közvetlen írás (WRITE DIRECT) privilegizált utasítások egyetlen byte-nyi adat átvitelére szolgálnak egy külső készülék és a főtár között, a csatorna felhasználása nélkül, mivel segítségükkel közvetlen adatút létesül a külső készülék számára.

Amikor a közvetlen vezérlés készülékének egy adatbyte-ot át kell vinnie, megszakítási jelet ad a hat külső vonal egyikére. Ez beállítja a megfelelő külső megszakítási regisztert, valamint külső megszakításkérés regiszterét. Ha a külső maszkbit (a PSW 7-es bitje) 1, a megszakítást elfogadják. A CPU a megszakítás elfogadásakor meghatározza a megszakítás forrását és átadja a vezérlést a közvetlen vezérlés mikroszubrutinjának, amely azután felhasználja mind a közvetlen olvasás-, mind a közvetlen írásutasításokat.

9.6.5.
Kezdeti-program betöltés

A kezdeti-program betöltés (**I**nital **P**rogram **L**oading: IPL) egy elsődleges hardware-betöltő („bootstrap: csizmahúzó"), amelynek feladata, hogy megindítsa az első program betöltését a főtárba. A kezdeti-program betöltőt a betöltőnyomógomb lenyomásával lehet indítani. A betöltőnyomógomb lenyomása után a következő lépéssorozat megy végbe:

1. A parancsot (olvasás, parancsösszefűzés, adatcím=0, leszámlálás=24) specifikáló csatornaparancsszó generálódik a főtár 0. helyén.
2. Egy B/K (I/O) start utasítás indul azon a készüléken (pl. kártyaolvasón), amelyet a betöltőkapcsoló megcímzett.
3. Ennek a B/K (I/O) start utasításnak a végrehajtása révén az első kártya első 24 oszlopa beolvasásra kerül a főtárba, a 0. helytől kezdve. Erre a kártyára lyukasztott információnak a következőket kell tartalmaznia:

1... 8 oszlop: az első PSW, amelyet a kezdeti-program betöltő befejezése után használni kell.
9...16 oszlop: a kezdeti-program betöltő parancslánc második csatornaparancsszava.
17...24 oszlop: a kezdeti-program betöltő parancsláncának harmadik csatornaparancsszava.

4. Az első csatornaparancsszó a parancsláncot szpecifikálja: így a következő csatornaparancszót a főtár 8. helyéről hívják le (vagyis a 9. kártyaoszlop). Ezért a további

működés már az éppen beolvasott információ vezérlése alatt megy végbe. (Az, hogy az első programot hogyan olvassák be, a csatornaprogramtól függ, amit a programozó ad meg.)

5. A kezdeti-program betöltés lezárul, amint a csatornaparancsszó többé nem specifikál parancsláncot. A főtár 0. helyén tárolt PSW-t betöltik a központi feldolgozóegységbe. A rendszer állapota az lesz, amit ebben a PSW-ben specifikáltak; az első utasítást ezen programállapotszóban levő utasításcímen jelzett helyről hívják le.

Feladatok

9.1. Fogalmazzuk meg és írjuk le CDL nyelven azt a vezérlőrendszert, amely a 9.4. ábrán látható három lehetséges hosszúságú utasításokat lehívja!

9.2. Fogalmazzunk meg és írjunk le CDL nyelven egy vezérlőrendszert, amely az utasításlehívó fázisban a regiszter- és tárcímzést valósítja meg, a 9.4. ábrán látható utasításformátumok feltételezésével!

9.3. Fogalmazzunk meg és írjunk le CDL nyelven egy struktúrát a 9.13. ábrán illusztrált tárvédelem realizálására! Használjuk fel a 9.4. ábrán bemutatott utasításformátumokat!

9.4. Vizsgáljuk felül a (8.32)...(8.38) utasításokban leírt prioritásos megszakítási rendszert, ha a megszakítási források és prioritási szintek az ebben a fejezetben leírtaknak felelnek meg!

9.5. Ismételjük meg a 9.4. feladatot arra az esetre, ha az öt megszakítástípushoz PSW átkapcsolást kell még pótlólag biztosítani!

9.6. Ismételjük meg a 9.5. feladatot arra az esetre, ha a PSW-ben megszakítási kódot kell még pótlólag beiktatni!

Irodalom

[1] AMDAHL, G. M.: *New Concepts in computing System Design.* Proceedings of the IRE, May, 1962, pp. 1073...1077.

[2] AMDAHL, G. M.—BLAAUX, G. A.—BROOKS, F. P., JR.: *Architecture of the IBM System/360.* IBM Journal of Research and Development, April, 1964, pp. 87...101.

[3] BLAAUX, G. S.—BROOKS, F. P., JR.—STEVENS, W. Y.—AMDAHL, G. M.—PADEGS, A.: *The Structure of System/360.* IBM Systems Journal 3, Nos. 2 and 3, 1964, pp. 118...196.

[4] FAGG, P.—BROWN, J. L.—HIPP, J. A.—DOODY, D. T.—FAIRCLOUGH, J. W.—GREENE, J.: *IBM System/360 Engineering.* Proceedings of the FJCC, Spartan Book Co., 1964, pp. 205...231.

[5] — —: *IBM System/360 Principles of Operation.* IBM Systems Reference Library, Form A22-6821-1, File No. S360-01.

[6] — —: *IBM System/360 System Summary.* IBM Systems Reference Libray, Form GA22-6810-10, File No. S360-00.

[7] — —: *A Programmer's Introduction to the IBM System/360 Architecture, Instructions, and Assembler Language.* IBM Corporation Form C20-1646-1.

10.
Csatornaszervezés

A korszerű számítógéprendszerek be-, ill. kimenetének (B/K) szervezése a főtár és a központi feldolgozóegység legjobb kihasználására törekszik olyan csatornarendszer mellett, amely egyetlen vagy több be-, ill. kimeneti műveletet hajt végre. A nehézség e kihasználás optimalizálásakor abból származik, hogy igen nagyok az eltérések a főtár és a be-, ill. kimeneti készülékek adatátviteli sebességei között. Például, egy korszerű számítógéprendszer főtárának adatátviteli sebessége elérheti az egy millió 64-bites szót másodpercenként, de ennél nagyobb is lehet, míg egy billentyűzet csak kb. tíz 6-bites szót tud percenként előállítani. Még a legnagyobb sebességű B/K készülék adatátviteli sebessége is csak kb. egy millió 8-bites byte másodpercenként. Ezért célszerű, hogy a CPU-ban a számítás és a feldolgozás a be-, ill. kimeneti adatátvitellel egyidejűleg történjék, és a be-, ill. kimeneti műveletekből származó megszakítások száma a gyakorlatilag elérhető legkisebb legyen. Ha csatornát alkalmazunk a be-, ill. kimenet vezérléséhez, az egyidejűség magasabb szinten valósítható meg. A csatorna megjelenése valóban az egyik legfontosabb szervezési előrelépést jelentette a modern számítógéprendszerekben.

Ebben a fejezetben először néhány be-, ill. kimeneti vezérlés szervezését mutatunk be, majd részletesen leírjuk az IBM System/360 40-es modelljének csatornaszervezését és működését. Ez a leírás a szelektorcsatornát, a multiplexcsatornát és a be-, ill. kimenetillesztést foglalja magában.

10.1.
A be-, ill. kimenet vezérlésének szervezése

A számítógéprendszerekben jelentősen megváltozott a be-, ill. kimenet vezérlésének szervezése az első elektronikus, tárolt programú számítógépekhez képest. Ennek részben az az oka, hogy a számítógépeket olyan alkalmazásokhoz használták, amelyek tárigénye jóval meghaladta a főtárkapacitást és így egy vagy több nagy kapacitású külső tárat kell használni be-, ill. kimeneti készülékként. A másik oka részben a

számítógép belső sebessége és a be-, ill. kimeneti készülékek sebessége közötti nagy különbözőségben keresendő.

A B/K vezérlés problémája lényegében a főtár és a be-, ill. kimeneti készülék közötti illesztés. Erre az illesztésre a különböző kódok, különféle formátumok, különböző sebességek, eltérő működésmódok és a különféle hardware-elemek miatt van szükség. A kódillesztést egyes esetekben a hardware, egyes esetekben a software kezeli, az adott be-, ill. kimeneti készüléktől függően. A sebesség- és formátumillesztésekhez rendszerint puffereket alkalmaznak, melyek segítségével egy tárbeli szót lehet összeszerkeszteni vagy széttagolni. A műveleti interface-t aszinkron működés és „tárciklus-lopás" alkalmazása jellemzi. Ennek az a célja, hogy a főtárat és a központi feldolgozóegységet a viszonylag lassú be-, ill. kimeneti adatátvitel ellenére is foglalkoztatni tudják. A hardware-illesztésre a különböző áramkörök paraméterei közötti eltérés és az eltérő megbízhatóság miatt van szükség, valamint azért, mert egy vagy több, különböző típusú be-, ill. kimeneti készüléket kell a számítógéprendszerhez csatlakoztatni.

Sokféle be-, ill. kimeneti vezérlésszervezési elvet dolgoztak ki és valósítottak meg. Ezek főleg a központi feldolgozóegység és a be-, ill. kimeneti készülékek közötti egyidejű működés módjában és fokában térnek el egymástól. Az egyidejű működés rendkívül fontos a B/K vezérlés szervezésében, mivel nagymértékben kihat a számítógéprendszer hatékony kihasználására.

10.1.1. Direkt programozott vezérlés

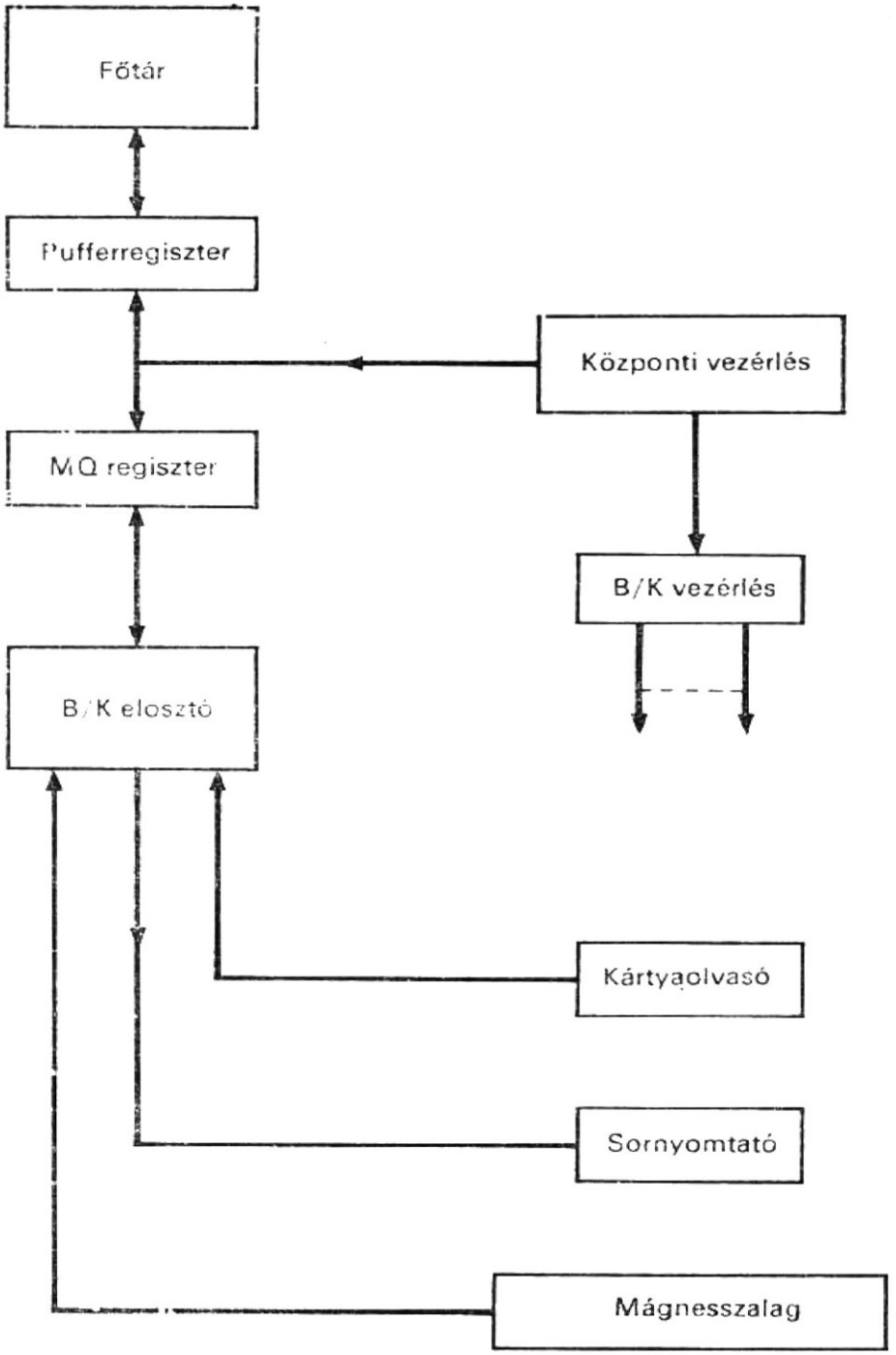
A direkt programozott vezérlés valamely be-, ill. kimeneti készüléktől jövő adategység átvitelét vezérli és ütemezi, egy be-, ill. kimeneti utasítás végrehajtása során. Többszörös adategység átviteléhez több utasítás szükséges. Például, a következő utasítások három szót olvasnak be valamelyik beviteli készülékről:

```
Olvass    x
Tárolj    a
Tárolj    b
Tárolj    c
```

ahol az *olvass* utasítás az x-szel specifikált be-, ill. kimeneti készülékről egy adategység beolvasását készíti elő, és elindítja a készülék mechanikai működését, ha az szükséges. Az egyes *tárolj* utasítások a beviteli készülékről érkező szót az a, b és c tárcímekre helyezik el.

A fenti átvitel először a beviteli készüléktől egy regiszterbe, pl. a központi feldolgozóegység MQ regiszterébe, majd ezután a főtártartalom regiszterén keresztül a főtárba történik (10.1. ábra). A be-, ill. kimeneti vezérlésre a CPU vezérlőegységét használják. Ez végzi a be-, ill. kimeneti készülék kiválasztását, a be-, ill. kimeneti művelet működésének szinkronizálását, az adatok átvitelét a főtárba, valamint a be-, ill. kimeneti készülék leválasztását, amikor a működés befejeződött. Ha az átviendő szavak számát is megadják a másolási utasításban, akkor ezt egy számláló tárolja. Minden egyes szóátvitelkor eggyel csökkentik a számlálót. Az átvitel akkor fejeződik be, amikor a számláló eléri a nullát.

Az előbbiek szerint egy szó átvitelére alkalmazott egy tárolási utasítás lehetővé teszi azt is, hogy különböző hosszúságú rekordokat lehessen átvinni különböző számú tárolási utasítással. Ez azzal az előnnyel is jár, hogy egy szalagrekord beolvasása különböző tárblokkokba tehát szétszórtan történhet. A többszörös tárolási utasításokat azonban rendszerint inkább egyetlen rövid iterációs hurokkal helyettesítik, amely egyetlen tárolási utasítást tartalmaz.



10.1. ábra
B/K direkt programozott vezérlés struktúrája

Amikor végrehajtanak egy tárolási utasítást, van arra idő, hogy több más utasítást is végrehajtsanak a be-, ill. kimeneti készülék lassú mozgása miatt. Például az alatt az idő alatt, amíg kártyaolvasás, kártyalyukasztás vagy sornyomtatás történik, kódkonverzió végezhető. A programozónak azonban ismernie kell a rendelkezésre álló idő mennyiségét, amely természetesen a be-, ill. kimeneti készülék típusától függ. Ha valamilyen adatátvitel során akkor érkezik be adat, amikor a következő tárolási utasítás még nem került végrehajtásra, a be-, ill. kimeneti készüléket leválasztják, és az adatátvitel megszakad. Minthogy a be-, ill. kimeneti programok elkészítése nem egyszerű, rendszerint rendszerprogramozói felhasználásra készülnek.

A direktprogramozott vezérlés a legegyszerűbb be-, ill. kimeneti vezérlés szervezési elv. A korai digitális számítógépekben, mint pl. az IBM 701 számítógépében használták, és még ma is alkalmazzák egyszerű digitális számítógépekben.

10.1.2 Puffertárak alkalmazása

Az előzőekben bemutatott direkt programozott vezérlésben a központi feldolgozóegység alkalmazása a be-, ill. kimeneti adatátvitel vezérlésére egyáltalán nem gazdaságos, mivel a CPU az ideje jelentős részét a be-, ill. kimenő adatokra való várakozással tölti. Ha egy külön, az adatok átmeneti tárolására (pufferezésére) szolgáló B/K (be-, ill. kimeneti) regiszterrel látják el, a CPU végrehajthatja a program egyéb utasításait, miközben a be-, ill. kimeneti adatátvitelre várakozik. A be-, ill. kimeneti regiszterek alkalmazásával tehát bizonyos fokig párhuzamos működés válik lehetővé, mivel a CPU-t fel lehet használni várakozás közben.

Hatékony CPU-működéshez olyan további eszközökkel kell ellátni a be-, ill. kimeneti készüléket, amellyel jelezni tudja a központi egységnek a be-, ill. kimeneti adatátvitel befejeztét. A korai számítógépekben ezt a végrehajtandó program egy vagy több helyére beiktatott, rövid iterációs hurokkal oldották meg. A legtöbb számítógép 1960 óta rendelkezik megszakítással; a be-, ill. kimeneti művelet működésének befejeztét tehát egy be-, ill. kimeneti megszakítási jel jelezheti. Ezekben a számítógépekben nincs szükség programhurokra.

Gyakorta nem elég hatékony a B/K regiszter alkalmazása az adatok átmeneti tárolására, mivel a várakozásra megengedett idő eléggé rövid. Egyetlen regiszter helyett egy regiszterkészletet vagy puffertárat használnak ilyenkor. A puffertárnak saját vezérlése van, ez szerkeszti össze vagy rakja szét az adatokat és ez tárol egy blokkot. Amikor a puffertár megtelik, a tartalmát egyszerre viszik át a főtárba.

A puffertár nagyobb kapacitása következtében kevesebbszer bontja meg a CPU-program végrehajtását. A puffer alkalmazása tehát gyorsabb működést tesz lehetővé a központi egység és a be-, ill. kimeneti készülék között. A puffertár alkalmazásának azonban két hátránya van:

— a blokkméret korlátozása, amely a puffertár kapacitásával függ össze; a blokkméret ui. nem lehet nagyobb a puffertár kapacitásánál, hacsak nem akarunk bonyolultabb vezérlést alkalmazni;

— a puffertár ára, amely jelentősen nőhet, ha több be-, ill. kimeneti készülékkel akarunk egyidejű működést biztosítani, mivel mindegyik működésben levő be-, ill. kimeneti készülékhez külön puffertár szükséges.

A puffertár kiküszöbölésére egy szellemes megoldást találtak, amit általában *ciklus-lopásnak* neveznek. A számítógép normál működésekor a CPU a tárat a program végrehajtásához főtárciklusok egymásutánjában foglalja le. Amikor valamely be-, ill. kimeneti adatátvitel lehetséges, a CPU „ellop" egy főtárciklust. A be-, ill. kimenő adatokat a főtárba vagy abból ez alatt a ciklusidő alatt viszik át és erre az időre a program végrehajtását felfüggesztik. E tárciklus végeztével a CPU ott folytatja működését, ahol a ciklusloPás előtt abbahagyta. A ciklusloPás mint művelet eltér a megszakítási művelettől, mivel egyrészt a megszakadás itt rögzített rövid időszakra szól, másrészt pedig nincs szükség a regiszterek tartalmának mentésére.

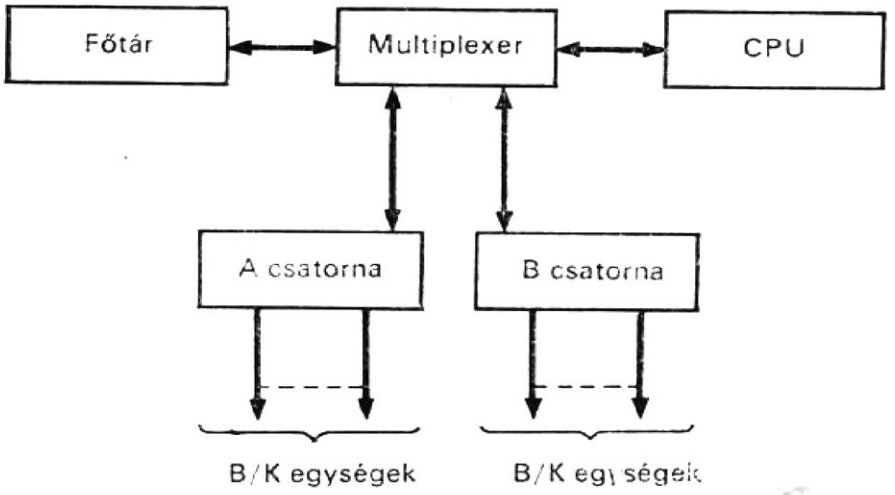
A ciklusloPás rendkívül hasznos, ui. a CPU nem veszít időt, eltekintve attól az időtartamtól, amelyre a be-, ill. kimeneti műveletben részt vevő szó lehívásához vagy tárolásához szükséges. Egy főtárbeli szóhoz átmeneti tárként elegendő egyetlen regiszter. A viszonylag olcsó tár egy része is használható pufferként. A módszer lényege az, hogy a ciklusloPás révén több be-, ill. kimeneti készülék működtethető párhuzamosan.

10.1.3. Adatcsatorna

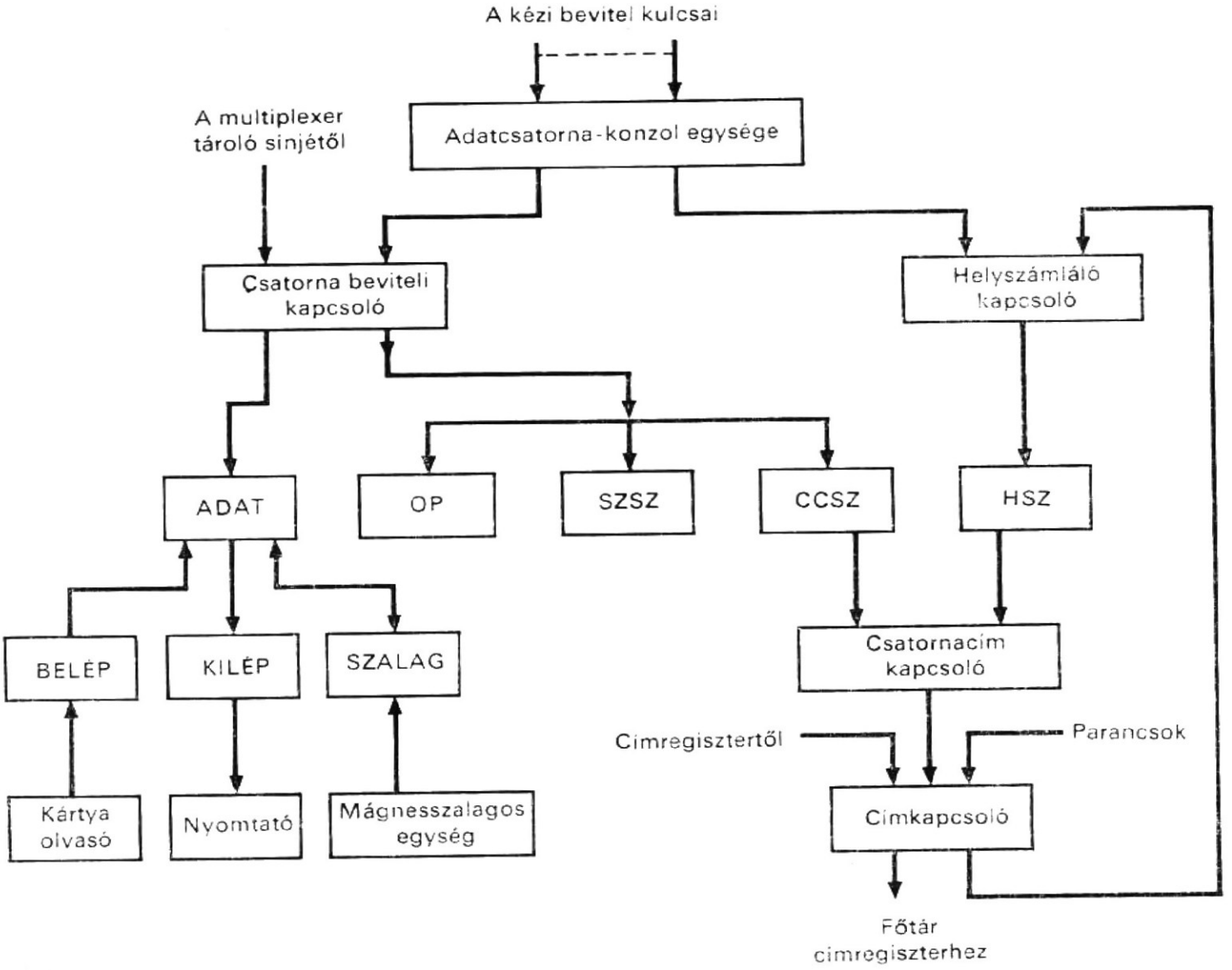
Az adatcsatornát fejlesztették ki annak a célnak elérésére, hogy nagymértékben egyidejű legyen a CPU és a be-, ill. kimeneti készülékek működése. A csatorna általánosan megfogalmazva be-, ill. kimeneti művelet kezelésére tervezett processzor. Amikor a CPU be-, ill. kimeneti utasítást hajt végre, megindítja a csatornát, és a be-, ill. kimeneti művelet vezérlését átadja a csatornának. Ezzel tehát nem maga végzi el a be-, ill. kimeneti művelet vezérlését. Ha egynél több csatorna van, több egyidejű be-, ill. kimeneti művelet mehet végbe. A központi egységhez hasonlóan az adatcsatorna is végre tud hajtani utasításokat, amelyeket *csatornaparancsnak* vagy egyszerűen parancsnak neveznek. E parancsok sorozata képezi a *csatornaprogramot*, amely sokoldalú be-, ill. kimeneti olvasási és írási műveleteket tesz lehetővé, a központi egység működésével egyidejűleg. Az adatcsatorna megjelenése az egyik legfontosabb szervezési előrelépést jelentette a korszerű digitális számítógéprendszerekben.

A továbbiakban példaként az IBM 7090 számítógép családjához tartozó IBM 7607 adatcsatornát fogjuk bemutatni. Itt összesen nyolc adatcsatorna csatlakoztatható a számítógéphez; tehát összesen nyolc be-, ill. kimeneti készülék tud egyidejűleg működni a központi egységgel. Amint a 10.2. ábrán látható, a csatornák és a központi egység egy kapcsolóegységen, az ún. multiplexéren keresztül csatolódik a főtárhoz.

Az IBM 7607 adatcsatorna konfigurációját a 10.3. ábra mutatja be. Az ADAT regiszter a főtár és valamelyik be-, ill. kimeneti készülék közötti pufferregiszter. Ez szolgál átmeneti tárolóként, amikor valamelyik tárbeli szót összerakják vagy szétbontják. Az OP műveleti (operációs) regiszter specifikálja a nyolc parancs egyikét. A CAC (**C**hannell **A**ddress **C**ounter) csatornacím-számláló (CCSZ) specifikálja a főtár és a be-, ill. kimeneti készülék között átviendő szavak számát. A HSZ helyszámláló az utasításszámláló funkciójához hasonlóan a pillanatnyi adatcsatorna-parancs helyét (plusz egyet) tartalmazza. Az SZBR regiszter a mágnesszalaghoz tartozó 36-bites puffertár. A 36-bites KBR a kártyaolvasó és a 36-bites NYBR a nyomtató pufferregiszterei. Ezeket egyesítve 72-bites regiszterként is lehet használni. A bemeneti kulcsok a csatornakonzolon elhelyezett azon kapcsolók, melyekkel kézi bevitel lehetséges. Ilyenek a 10.3. ábrán látható: címkapcsoló, beviteli kapcsoló és számlálókapcsoló és a jelutakat biztosító kapcsolóegységek. A struktúrát az alábbi utasításokkal írjuk le.



10.2. ábra
Az IBM 7090 számítógéprendszer felépítése



10.3. ábra
Az IBM 7607 adatcsatornájának felépítése

```
Comment, az IBM 7607 adatcsatorna struktúrája                    (10.1)
Register,  ADAT(S,1—35),          $adatregiszter
           OP(S,1,2,19),          $műveletvezérlő regiszter
           SZSZ(3—19),            $szószámláló
           CCSZ(3—17),            $csatornacím-számláló
           HSZ(3—17),             $helyszámláló
           SZBR(S,1—35),          $szalagregiszter
           NYBR(S,1—35),          $kiviteli regiszter
           KBR(S,1—35),           $beviteli regiszter
Switch,    KBR KULCS(S,1—35), (BE, KI), $beviteli kulcsok az adatcsatorna-kon-
                                        zolon
```

Az IBM 7607-es adatcsatorna megindulása után aszinkron üzemben dolgozik. Egyetlen parancs egy nagy adatblokkot vihet át a főtár és a be-, ill. kimeneti készülék között, ezzel egyidejűleg a főprogramban egy sor utasítás kerülhet végrehajtásra.

Bár több be-, ill. kimeneti készülék kapcsolódhat egy adatcsatornához, egyszerre csak egy lehet működésben, mivel az adatcsatorna csak egy be-, ill. kimeneti készülék adatátvitelét tudja ellátni egyidőben. Ha egynél több adatcsatorna kiszolgálását kell elvégezni, a csatornákhoz egy prioritási rendszert kell készíteni, ez biztosítja, hogy a kívánt sorrend szerint lehessen őket kiszolgálni.

Az IBM 7607 adatcsatornát két utasítás végrehajtásával lehet elindítani a központi egységben. Pl. ahhoz, hogy az Y, Y+1, ..., Y+49 helyeken levő, 50 főtárbeli szóból álló blokkot beírassuk az A csatorna 2. egységén levő mágnesszalagra, a következő utasításokra és parancsokra van szükség:

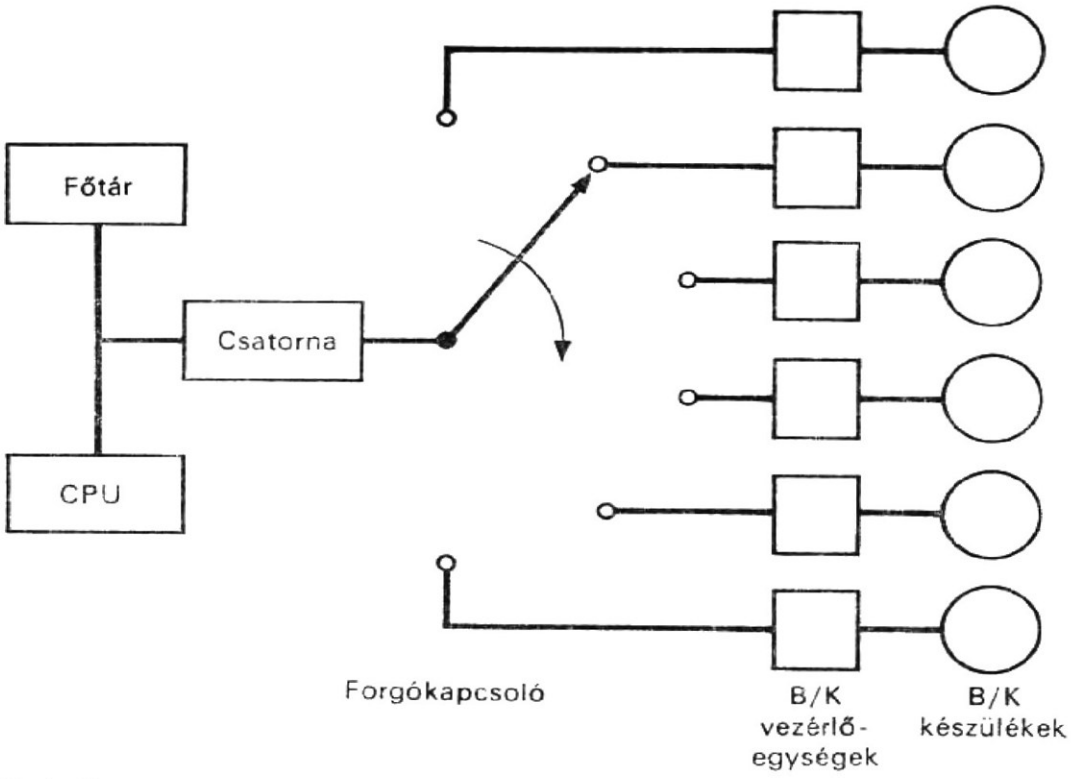
```
        WTBA  2
        RCHA  JOE
        ..........
        ..........
JOE     IOCD  Y,,50
```

A WTB utasítás (szalagra bináris formában történő felírás) utasítás kiválasztja az A csatornán a 2. mágnesszalagegységet és vezérli a bináris felírási műveletet. Az RCH utasítás beállítja és betölti az A csatornát és megadja az A csatornának azt a JOE tárcímet, ahol az IOCD parancs van. Az IOCD parancs (bevitel és kivitel számlálással és leválasztás) az Y, Y+1, ..., Y+49 helyeken levő szóblokkot felíratja az A2 mágnesszalag egységre. Amikor beírásra került az 50 szó, a rekord-vége jel íródik be és a csatorna leválasztja a mágnesszalag egységet.

10.1.4. Multiplexcsatorna

Amint az előzőekben leírtuk, egy adatcsatorna egyetlen be-, ill. kimeneti műveletet tud a központi egységgel egyidőben végrehajtani. Az ilyen típusú csatornát *szelektorcsatornának* hívják. A szelektorcsatorna nagy sebességű be-, ill. kimeneti adatátvitelt tesz lehetővé, azonban lassú be-, ill. kimeneti készülék használata esetén gazdaságtalan, mivel egy adatcsatorna egyetlen készüléket működtethet egyidőben. Rendkívül előnyös, ha a csatorna egyidejűleg több lassú be-, ill. kimeneti készüléket tud működtetni. Ilyen problémák merültek fel korábban a híradástechnikában, és a megoldás az időmultiplexelés lett. A csatornaműködés időbeli multiplexelésének alapgondolatát a 10.4. ábra illusztrálja. Amint azon látható, hat be-, ill. kimeneti vezérlőegységet alkalmazunk, mindegyik vezérlőegység egyetlen be-, ill. kimeneti készüléket vezérel. A hat vezérlőegység forgókapcsoló — vagy ezt megvalósító elektronika — segítségével osztja fel a csatorna idejét. A csatorna a forgókapcsoló megfelelő helyzetbe fordulásával kezdeményezi vagy látja el a be-, ill. kimeneti műveletet, egyik vezérlőegység után a következőre kapcsolódva. Az időmultiplexelés megengedhető, mivel bőséges idő áll rendelkezésre a várakozáshoz, míg egy be-, ill. kimeneti készülék egy műveletet végrehajt. Az ilyen típusú csatornát *multiplexcsatornának* hívják.

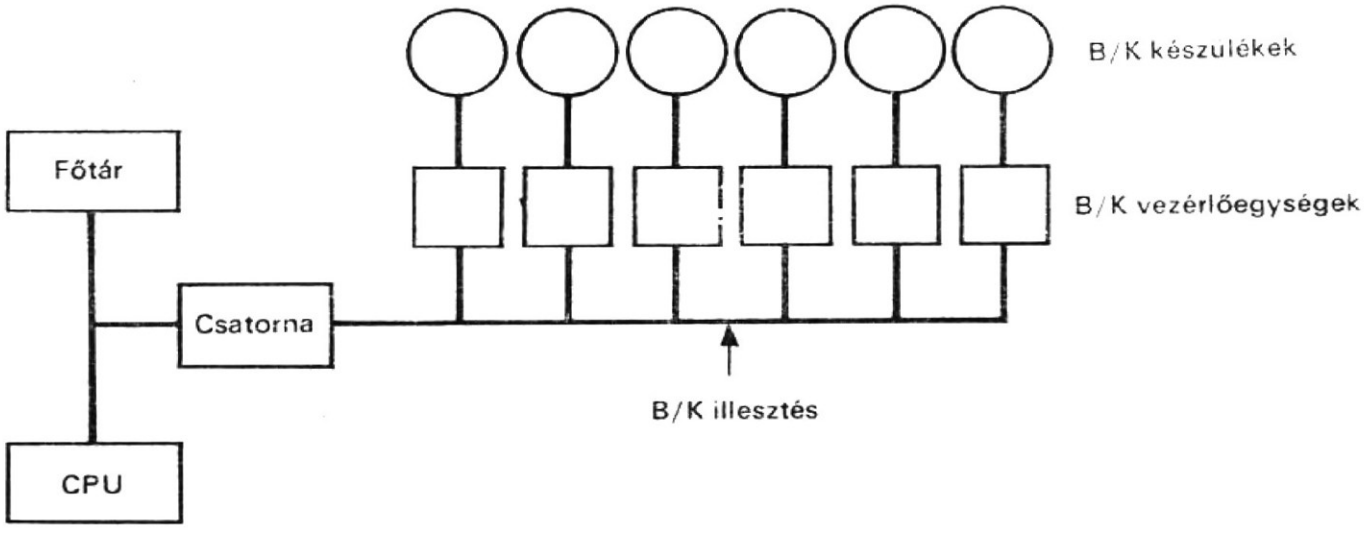
A 10.4. ábrán bemutatott multiplexcsatorna-elrendezés gyakorlati megvalósítása nem célszerű. Ennél a vezérlőegységek csatlakoztatásához annyi kábel szükséges, amennyi a be-, ill. kimeneti vezérlőegységek száma. Előnyösebb megoldás látható a 10.5. ábrán, ahol csak egyetlen kábel köti össze a be-, ill. kimeneti vezérlőegységeket a csatornával. Ezt a sínrendszert, ill. annak képzeletbeli síkkal való metszetét metszősíknak, angolul interface-nek neveznek. Ez egyéb vezetékeken kívül, címsíneket, ill. vezetékeket és letapogató vezetékeket tartalmaz. Az egyes be-, ill. kimeneti vezérlőegységeket egy-egy cím jelöli ki. A csatorna jelet küld a letapogató vezetékre, amelyet csaknem azonnal követnek a címvezetékekre küldött jelek. Amikor a be-, ill. kimeneti vezérlőegységet kiválasztja a letapogató jel, az összehasonlítja a bejövő címet a kijelölt címmel. Ha a két cím nem egyezik, a letapogató jel továbblép a következő be-, ill. kimeneti készülékhez, egészen addig, amíg meg nem találja a megegyező címmel ellátott egységet, vagy amíg a letapogató jel vissza nem tér a csatornához, je-



10.4. ábra
Az adatcsatorna multiplexelése

lezve, hogy a kért be-, ill. kimeneti vezérlőegység vagy nem működőképes, vagy nincs a csatornához csatolva.

A multiplexcsatorna valójában több alcsatornát tartalmaz. Pl. az IBM System/360 40-es modelljének multiplexcsatornájában összesen 128 alcsatorna lehet. Minden alcsatorna tartalmazza az összes szükséges információt ahhoz, hogy valamelyik be-, ill. kimeneti készüléket vezérelje. Amikor különböző alcsatornák szimultán hajtanak végre be-, ill. kimeneti műveleteket, azt mondjuk, hogy a multiplexcsatorna *multiplex üzemmódban* dolgozik. Ebben az üzemmódban az egyes be-, ill. kimeneti egységek logikailag csak annyi időre kapcsolódnak a csatornához, amennyi idő egy adatbyte átviteléhez szükséges; ebből az okból, a multiplex üzemmódot gyakran *byte-üzemmódnak* is nevezik. Két byte átvitele között a B/K vezérlőegységet leválasztják, és a csatorna felszabadul további B/K vezérlőegységek működtetésére. Ezért a multiplexcsatornán levő különböző be-, ill. kimeneti egységekből jövő adatátviteli folyamatok be



10.5. ábra
A multiplexcsatorna felépítése

vannak ékelve egymásba (interleaved). A csatorna működtethet egyetlen be-, ill. kimeneti készüléket is, de ezt nagy sebességgel. Ebben az esetben azt mondjuk, hogy a csatorna „szelektor" vagy lökésszerű üzemmódban, vagy az angol elnevezés szerint burst módban van. Amikor a multiplexcsatorna burst-üzemmódban van, szelektorcsatornához hasonlóan működik. A csatorna multiplexelésének következménye a csatorna adatátviteli sebességének csökkenése.

Mind a multiplexcsatorna, mind a szelektorcsatorna a következő funkciók valamelyikét vagy mindegyikét tudja megvalósítani:

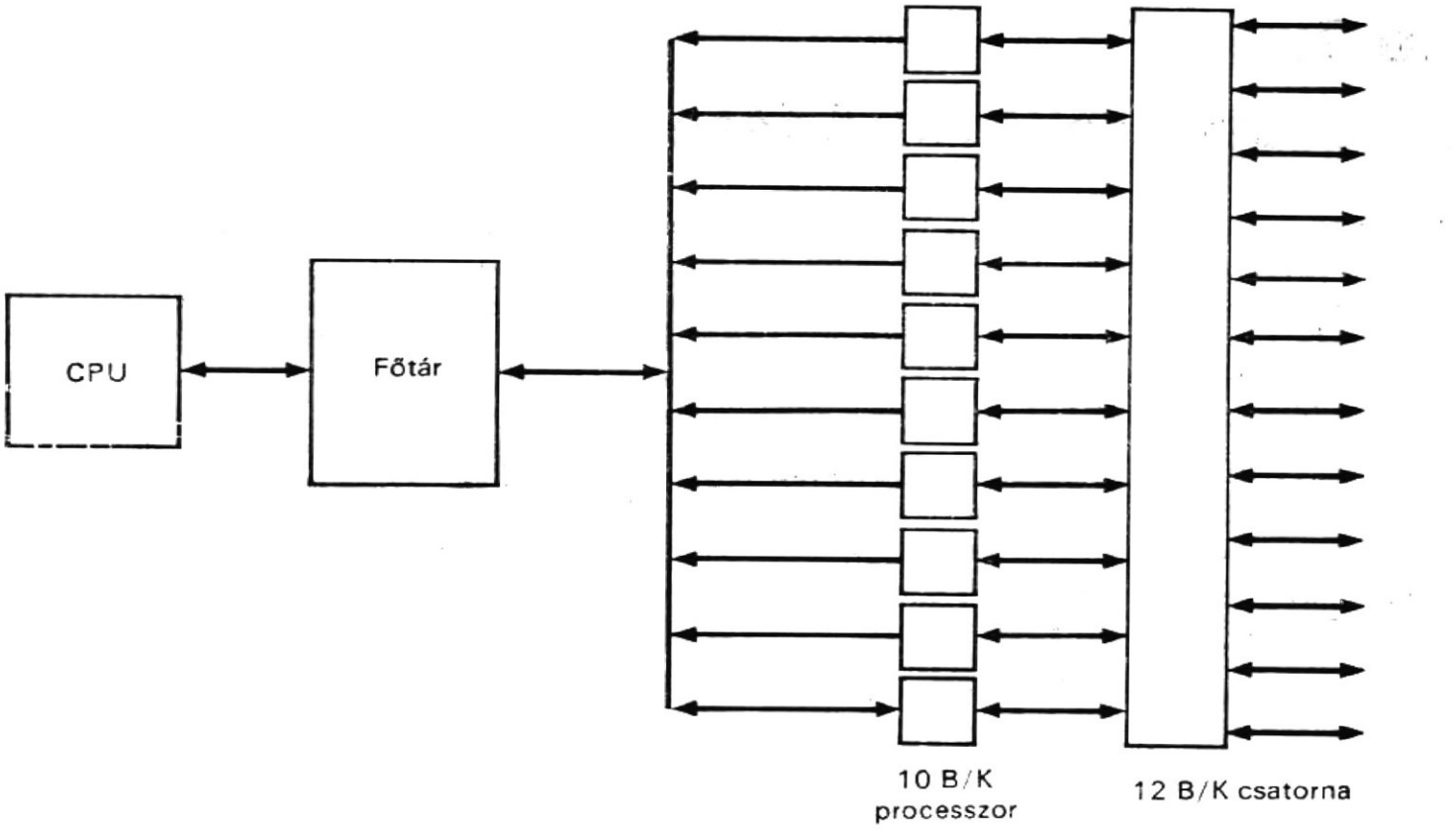
1. A be-, ill. kimeneti utasítás fogadása a CPU-tól.
2. A be-, ill. kimeneti utasítás által specifikált be-, ill. kimeneti készülék megcímzése és kiválasztása.
3. A csatornavezérlési információ lehívása, dekódolása, tesztelése és végrehajtása.
4. Vezérlőjelek előállítása a be-, ill. kimenetillesztés működtetésére.
5. Az állapot- (státusz-) információ fogadása és elhelyezése a főtárban.
6. Az adatok átmeneti tárolása a főtárba vagy abból történő adatátvitelhez.
7. Az adatbyte-ok leszámlálása és párosság ellenőrzése.
8. A csatornaállapot-információ fenntartása.
9. A be-, ill. kimeneti készülékektől jövő megszakításkérések kezelése.
10. A be-, ill. kimeneti megszakítás jelzése a CPU-nak.

10.1.5. A be-, ill. kimeneti processzorok

Külön be-, ill. kimeneti (B/K) processzorokat használnak abból a célból, hogy a központi feldolgozóegységben minél kevesebb legyen a be-, ill. kiviteli vezérlés, tehát minél több számítást és feldolgozást végezzen. A B/K processzor tárolt programú számítógép, amely rendszerint a CPU-tól függetlenül működik. Az adatcsatornák ebben az esetben a B/K processzorokhoz kapcsolódnak. Különböző típusú B/K processzorokat használnak. Lehetséges, hogy a B/K processzornak megfelelő aritmetikai, logikai, elágaztatási és pufferezési képessége van a be-, ill. kimeneti programok végrehajtására. Felépíthető úgy is a rendszer, hogy a B/K processzor nem hajt végre komplex műveleteket, hanem a CPU-t hívja be a feldolgozás elvégzésére. A B/K processzorok, a be-, ill. kimenő adatokkal végzett be-, ill. kimeneti műveletek után a főtárban helyezik el az adatokat, és azután megszakítás segítségével értesítik erről a központi egységet.

A továbbiakban a CDC 6000 számítógépcsaládot használjuk példaként, amelynek a központi feldolgozóegységen és a 131 072 szó (60-bites szóhosszú) kapacitású főtáron kívül tíz B/K processzora (ezeket perifériás vagy vezérlőprocesszoroknak nevezik) és 12 B/K csatornája van (10.6. ábra). Az egyes perifériás processzoroknak (vagy rövidítve, PP) 4096 12-bites szóból álló táruk van, és bármelyik a 12 csatorna bármelyikéhez hozzá tud férni. A PP egy egyszerű számítógép, amely mind a CPU-tól, mind a többi PP-től függetlenül tud működni. A CPU és valamennyi PP hozzá tud férni a főtárhoz. Bár a központi egység igen gyors processzor, nem tud be-, ill. kimeneti műveletet végezni. Ha a CPU-nak be-, ill. kimeneti műveletre van szüksége, az adatokat a főtárba helyezi el és jelet küld a PP-hez. A PP fogadja a jelet, és a központi egység más feldolgozással folytatja munkáját. Ha ez kiviteli művelet, a PP az adatokat átviszi a saját tárába, és azt követően a csatornán keresztül valamelyik be-, ill. kimeneti készülékhez.

A 10 PP mindegyike egyforma. Mindegyiküknek van saját, 4096-szavas tára, címregisztere, akkumulátora és vezérlőregiszterei. Egyébként, a tíz PP időosztásban egyet-



10.6. ábra
A CDC 600 számítógépcsalád B/K processzorai

len feldolgozóegységen osztozik. A PP 64 utasítással rendelkezik, ezek az összeadásra, kivonásra, a logikai, elágaztató, be-, ill. kiviteli műveletekre, a főtárhoz való hozzáférésre és a központi feldolgozóegység elérésére használhatók. A PP meg tudja állapítani, hogy melyik csatorna nem aktív, és ki tud választani egyet arra, hogy egy vagy több szót átvigyen egy utasítással. Amikor a PP be-, ill. kimeneti műveletet kezdeményez, rendszerint a programhurokban marad mindaddig, amíg a művelet be nem fejeződik, mivel a CDC 6000-es számítógépekben nincs megszakítási hardware.

A rendszerben 12 B/K csatorna van, és mindegyik csatorna egy 12-bites csatornaregisztert és két jelzőbitet (flag-et) tartalmaz. A B/K csatornákon végzett közvetlen tevékenységhez 12 utasítás áll rendelkezésre. Az egyes csatornák átviteli sebessége másodpercenként egy millió 12-bites szó. Mindegyik csatornához egy vagy több be-, ill. kimeneti készülék csatlakozhat. Egyidőben egy csatornával csak egy készülék kommunikálhat, de a 12 csatorna egyszerre lehet aktív.

A perifériás processzorok felügyelő- (supervisor) processzorként is használhatók. Pl. az egyik, vezérlő (executive) PP-nek kijelölt processzor tartalmazhatja a törzs (master) vezérlőprogramot. Ez ütemezi az összes munkát, beleértve a CPU által végrehajtott munkákat is. Egy másik PP, amelyet monitorprocesszornak neveznek ki, a programok végrehajtását figyeli a CPU-ban. A fennmaradó nyolc PP-t egy közös alapba (pool-ba) fogják össze. Szükség esetén a vezérlő PP választ ki egyet a közös alapból, és ezt a PP-t utasítja, hogy hajtsa végre a kívánt műveletet.

10.2. A csatorna működése

A csatorna a számítógéphez kapcsolt be-, ill. kimeneti készülékek vezérlésére szolgáló berendezés. Ez vezérli az adat és vezérlési információ átvitelét a be-, ill. kimeneti készülékek és a CPU, valamint a központi egység és a főtár között. Szeretnénk néhány dolgot bemutatni a korszerű csatornákról, ezért e fejezet további részeiben az IBM System/360 40-es modelljének csatornaszervezését ismertetjük.

Az IBM System/360 számítógépek csatornaműködésében öt vezérlőszó vesz részt:

1. CCSZ — csatornacímszó (**C**hannel **A**ddress **W**ord: CAW),
2. CPSZ — csatornaparancsszó (**C**hannel **C**ommand **W**ord: CCW),
3. CSSZ — csatornaállapot- (státusz) szó (**C**hannel **S**tatus **W**ord: CSW),
4. EVSZ — egységvezérlő szó (**U**nit **C**ontrol **W**ord: UCW),
5. PSSZ — programállapot- (státusz) szó (**P**rogram **S**tatus **W**ord: PSW).

A következőkben a felsorolt vezérlőszavakat írjuk le, valamint a csatornaparancsokat, a csatornaprogramot, a be-, ill. kimeneti (B/K) utasítások végrehajtását és a be-, ill. kimeneti (B/K) megszakítások kezelését.

10.2.1. A csatornacímszó

A csatornacímszó egész hosszúságú, tehát 32 bites szó, amely állandóan a főtár 72. helyén tárolódik. Formátuma a 10.7. ábrán látható. Három mezőre oszlik:

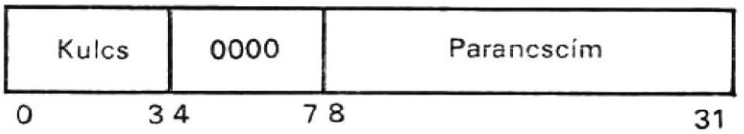
1. A kulcsmező (0...3 bit) a B/K tárvédelmi kulcsot tartalmazza, amely a főtárhoz való hozzáférés engedélyezését vezérli a be-, ill. kimeneti művelet során.
2. A nullamező (4...7 bit) mindig nullákat kell, hogy tartalmazzon.
3. A csatornaparancs-címmező (8...31 bit) határozza meg az első csatornaparancsszó helyét a főtárban.

A CCSZ (CAW)-nek a főtár 72. helyén kell lennie, mielőtt a B/K start (START I/O) utasítás végrehajtásra kerül. Amikor a B/K start (START I/O) utasítás végrehajtódott, a CCSZ szót lehívjuk a tárból. Az első csatornaparancsot ezután a CCSZ-ben meghatározott parancscímről hívják le.

10.2.2. A csatornaparancsszó

A csatornaparancsszó (CPSZ) a főtárban tárolt kettős szó. A formátumot a 10.8 ábrán láthatjuk. Öt mezőre oszlik:

1. A parancskódmező (0...7 bit) a hat B/K parancs egyikét specifikálja, amelyeket a 10.1. táblázatban soroltunk föl.



10.7. ábra
Csatornacímszó (CCSZ) formátuma

bit	
0—7	Parancskód
8—31	Adatcím
32—36	Jelzőbitek
32	Adatláncolást jelző bit
33	Parancsláncolást jelző bit
34	Hosszjelzés-elnyomást jelző bit
35	Ugrást (skip) jelző bit
36	Programvezérelt megszakítást jelző bit
37—39	Nulla
40—47	Nincs felhasználva
48—63	Számláló

10.8. ábra
Csatornaparancsszó (CPSZ) formátuma

2. Az adatcímmező (8...31 bit) azt az első főtárhelyet határozza meg, ahol az első adatszó tárolandó, olvasás, olvasás vissza és érzékelés műveletek esetén. Továbbá ez specifikálja azt az első főtárhelyet, ahová az első szót kell helyezni beírási és vezérlési művelet esetén.

3. A jelzőbitmező (32...36 bit) öt jelzőbitet (flaget) tartalmaz:

— a CS *adatláncolást jelző bit* (32. bit), amely 1 értéke esetén adatláncolást ír elő (l. a következőkben),

— a CC *parancsláncolást jelző bit* (33. bit), amely 1 értéke esetén parancsláncolást jelez (l. a következőkben),

— az SLI *hosszjelzés-elnyomást jelző bit* (34. bit), amely 1 értéke esetén elnyomja a helytelen hosszjelzést, amennyiben a csatorna helytelen hossz-specifikációt észlel,

— az *ugrást* (skip) *jelző bit* (35. bit), amely 1 értéke esetén elnyomja a beviteli egységtől a főtárhoz történő adatátvitelt, miközben a külső eszköz tovább mozog (ilyen pl. kártyaoszlopok átugrása). Ezt a jelzőbitet olvasási, visszafelé olvasási és érzékelési műveletekhez használják,

— a PCI *programvezérelt megszakítást jelző bit* (36. bit), amely 1 értéke esetén be-, ill. kimeneti megszakítást állít elő és ezután státusz információt szolgáltat a központi egységnek.

4. A nullamező (37...39 bit), amely mindig nullákat kell, hogy tartalmazzon, kivéve, ha TIC parancsról van szó.

5. A számlálómező (48...63 bit) határozza meg az átvitelre kerülő byte-ok számát.

A 40...47 bitet nem veszi figyelembe a csatorna.

10.1. táblázat

Csatornaparancskód

Parancs	Kód
Érzékelés (előre olvasás esetén)	MMMM 0100
Átvitel a csatornán	XXXX 1100
Visszafelé olvasás	MMMM 1101
Írás (előre)	MMMM MM01
Olvasás (előre)	MMMM MM10
Vezérlés	MMMM MM11

M a módosítóbiteket; X az elhanyagolt biteket jelöli.

10.2.3.
A csatornaparancsok

Amint a 10.1. táblázatból látható, hat B/K (be-, ill. kimeneti) parancs van: érzékelés, átvitel a csatornában (**T**ransfer-**I**n-**C**hannel: TIC), visszafelé olvasás, írás, olvasás, vezérlés. Az írás műveletkor a csatorna szavakat hív le a főtárból, és egyszerre egy byte-ot továbbít át a be-, ill. kimeneti készülékhez. Az olvasás vagy a visszafelé olvasás műveletkor a csatorna egyszerre egy byte-ot fogad a be-, ill. kimeneti készüléktől, és egyidőben egy szót továbbít a főtárhoz. Az olvasási műveletben a főtárban a szavakat a címek növekvő sorrendjében helyezik el. A visszafelé olvasás műveletben a szavakat a főtárban a címek csökkenő sorrendjében tárolják. Az érzékelés műveletkor a megcímzett be-, ill. kimeneti készülék a pillanatnyi állapotát (státuszát) közli, és a rendkívüli körülmények a megszakítási rendszeren keresztül kerülnek kiértékelésre. Az átvitel a csatornában műveletkor e parancs adatcímmezeje által specifikált helyről kerül lehívásra a CPSZ (a csatornaparancsszó) a sorrendben következő CPSZ helyett. Tehát ez az utasítás elágazást hoz létre, és egyik CPSZ sorozatról egy másikra lehet áttérni.

10.2.4.
A csatornaprogram

A csatornaprogram B/K utasítással specifikálja a végrehajtandó műveleteket, és egy vagy több csatornaparancsszó sorozatából áll. Ha egynél több csatornaparancsszót tartalmaz, azt mondjuk, hogy a CPSZ-ek láncoltak. A láncolásnak két típusa lehet:

— parancsláncolás és
— adatláncolás.

A *parancsláncolás* több különböző típusú csatornaparancsszóra vonatkozik, míg az adatláncolás több azonos CPSZ-t jelent, amelyeknek azonban különbözőek az adattartományaik.

A csatorna a parancsláncolással az új csatornaparancsszót használja a be-, ill. kimeneti készüléken végzendő új művelet végrehajtására. Ezzel a központi egység számára lehetővé válik pl. olyan sorozatok iniciálása, mint pl. a többszörös nyomtatás vagy többszörös szalagblokk olvasása egyetlen B/K utasítással. A parancsláncolás révén a csatorna több B/K utasításból álló csatornaprogramot tud végrehajtani.

Az *adatláncolás* esetén a csatorna az új csatornaparancsszót arra tudja használni, hogy más adattartományt jelöljön ki az eredeti be-, ill. kimeneti műveletekhez, de a be-, ill. kimeneti készülék ezt a műveletet folytatja. Tehát az adatláncolás az adat átszervezését teszi lehetővé, mialatt a főtár és a be-, ill. kimeneti készülék között átvitelre kerül.

Három egyszerű csatornaprogramot mutatunk be. Az első példa mindössze egyetlen csatornaparancsszót tartalmaz:

Parancs	Adatcím (hexadecimális)	Jelzőbitek (bináris)	Számláló (decimális)
Olvas	1000	00000	200

Ez a csatornaprogram 200 byte-ból álló teljes rekordot olvas a mágnesszalag egységről, amelyet a módosítóbitek specifikálnak a parancskódban a főtár hexadecimális 1000-től 10C8-ig terjedő helyeire.

A második példa a következő három csatornaparancsszóból (CPSZ) áll:

CPSZ	Parancs	Adatcím (hexadecimális)	Jelzőbitek (bináris)	Számlálás (decimális)
1	Olvas	1000	10000	50
2	Elhanyagolva	Elhanyagolva	10010	20
3	Elhanyagolva	2000	00000	130

Ez a csatornaprogram egy 200 byte-os rekordot olvas ki az egyik mágnesszalag egységről (amelyet a módosítóbitek specifikálnak a parancskódban), mégpedig az 1...50 byte-ot a főtárban, a hexadecimális 1000 helytől kezdve helyezi el, az 51...70 byte-ot nem veszi figyelembe, és a 71...200 byte-ot ismét a főtárban helyezi el, a hexadecimális 2000 helytől kezdve. Az olvasási művelet láncolással jött létre, mivel a CD jelzőbit értéke az első és a második csatornaparancsszóban 1.

A harmadik példa a következő három CPSZ-ből áll:

CPSZ	Parancs	Adatcím (hexadecimális)	Jelzőbitek (bináris)	Számláló (decimális)
1	Írás	1000	01000	50
2	Vezérlés	Elhanyagolva	01100	1
3	Vezérlés	Elhanyagolva	00100	1

Ez a csatornaprogram egy 50 byte-os utócimkét ír a mágnesszalagra (amelyet a parancskódban levő módosítóbitek specifikálnak) a főtár hexadecimális 1000 helyétől kezdődően, szalagjelzést ír be (amelyet a parancskódban levő módosítóbitek specifikálnak) és visszacsévéli és kiüríti a mágnesszalagot (amit a parancskódban levő módosítóbitek specifikálnak). Az írási művelet parancsláncolással jön létre, mivel a CC jelzőbit értéke az első és a második csatornaparancsszóban 1. Minthogy a csatornaparancsszó kezdeti számláló értéke sosem lehet nulla, kezdeti nemnulla számláló állást kell specifikálni azokban a vezérlőparancsokban, amelyek nem végeznek vezérlő adatátvitelt, és az SLI jelzőbitet kell beállítani arra, hogy elnyomja a helytelen hossz jelzést.

Megfelelő csatornaprogrammal a következő be-, ill. kimeneti adatátviteli módokat és művelettípusokat lehet megvalósítani:

— nem folytonos olvasás (egy fizikai rekordnak a tár több, nem folytonos területére történő beolvasása),

— kiemelés (csak a rekord kiválogatott részeinek beolvasása a tárba),

— adatnélküli be-, ill. kimeneti műveletek (mint pl. visszaléptetés és visszacsévélés) és

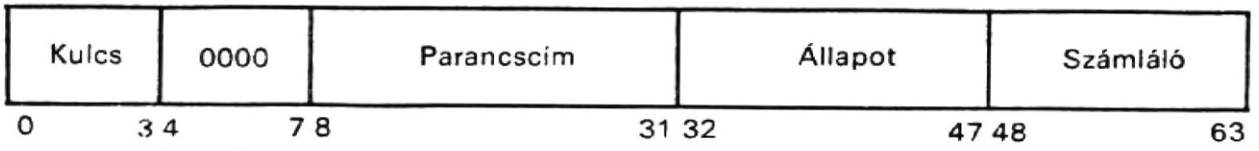
— műveletsorozat ugyanazon a készüléken (pl. olvasás blokk-közön keresztül).

10.2.5. A csatornaállapot-szó

A csatorna, a B/K vezérlőegység, a be-, ill. kimeneti készülék állapotát, valamint a következő csatornaparancsszó helyét a CSSZ csatornaállapot-szó határozza meg. Ez egy kettős szó, amelyet a főtár 64. helyén tárolnak. A csatornaállapot-szót a be-, ill. kimeneti művelet befejezte után felfrissítik, hogy információt adjon a B/K befejezés állapotáról.

A csatornaállapot-szó (CSSZ) formátuma a 10.9. ábrán látható. Öt mezeje van:

1. A tárvédelmi kulcs mező (0...3 bit) tartalmazza a csatornához tartozó védelmi kulcsot.



0—3 Védelmi kulcs
4—7 Nulla
8—31 Parancscím
32—47 Állapot
32 Figyelem
33 Állapotmódosító
34 Vezérlőegység vég
35 Foglalt
36 Csatorna vég
37 Készülék vég
38 Készülék vizsgálat
39 Különleges feltétel
40 Programvezérelt megszakítás
41 Helytelen hossz
42 Programellenőrzés
43 Védelemellenőrzés
44 Csatornaadatok ellenőrzése
45 Csatornavezérlés ellenőrzése
46 Illesztésvezérlés ellenőrzése
47 Láncolásellenőrzés
48—63 Számláló

10.9. ábra
Csatornaállapotszó (CSSZ) formátuma

2. A nullamező (4...7 bit) mindig nulla.
3. A parancscímmező (8...31 bit) tárolja a következő csatornaparancsszó címét.
4. Az állapotmező (32...39 bit) első része a B/K vezérlőegység és a be-, ill. kimeneti készülék állapotát tárolja, míg a második része (40...47 bit) a csatorna állapotát tárolja.
5. A számlálómező (48...63 bit) a rezidens számláló értékét még átvinni kívánt információegységek számát adja meg a be-, ill. kimeneti művelet befejezése után.

A 32...39 állapotbitek jelentése a következő:

A 32. bit *(figyelem)* akkor állítódik elő, amikor kézi műveletet kezdeményeznek, pl. konzolírógépen keresztül.

A 33. bit *(állapotmódosító)* azt jelzi, hogy a be-, ill. kimeneti készülék nem tudja végrehajtani a parancsot.

A 34. bit *(vezérlőegység vég)* jelzi, ha a vezérlőegység éppen szabad.

A 35. bit *(foglalt)* a készülék foglaltságát jelöli.

A 36. bit *(csatorna vég)* akkor jön létre, ha a B/K vezérlőegységnek már nincs szüksége a csatornára, és a be-, ill. kimeneti műveletet saját maga is be tudja fejezni.

A 37. bit *(készülék vég)* mindannyiszor előállítódik, ahányszor a készülék eléri mechanikai végpontját, vagy ha a készülék a *nincs kész* állapotból a *kész* állapotba kerül (pl. egy mágnesszalaghajtás).

A 38. bit *(készülék vizsgálat)* rendszerint akkor jön létre, amikor a vezérlőegység vagy a készülék géphibát fedezett fel.

A 39. bit *(különleges feltétel)* akkor generálódik, ha a be-, ill. kimeneti készülék olyan állapotot fedez fel, amely normális esetben nem fordul elő (pl. szalagjelzés felismerése).

A 40...47 állapotbitek jelentése a következő:

A 40. bit (programvezérelt megszakítás — **P**rogram **C**ontrolled **I**nterrupt: PCI) akkor generálódik, ha a csatornaállapot-szó a csatornaparancsszóban levő PCI jelzőbit miatt kerül felfrissítésre.

A 41. bit *(helytelen hossz)* akkor jön létre, ha a csatornaparancsszó számlálója nem nulla a be-, ill. kimeneti művelet végén, és az SLI jelzőbit 0.

A 42. bit *(programellenőrzés)* érvénytelen állapotot jelöl, mint pl. érvénytelen csatornaparancsszó címet vagy érvénytelen tárvédelmi parancs kulcsot.

A 43. bit *(védelem-ellenőrzés)* akkor generálódik, ha a be-, ill. kimeneti művelet közben kísérlet történik a főtárvédelem megsértésére.

A 44. bit *(csatornaadat-ellenőrzés)* azt jelzi, hogy a csatorna érvénytelen adatot fedezett fel.

A 45. bit *(csatornavezérlés-ellenőrzés)* bármilyen, a csatornavezérlést érintő géphiba esetén generálódik.

A 46. bit *(interface-vezérlés-ellenőrzés)* az illesztésen megjelenő érvénytelen jeleket mutatja ki.

A 47. bit *(láncolás-ellenőrzés)* túlfutási viszonyokat jelez, a beviteli műveleteken végzett adatláncolás közben.

10.2.6. Az egységvezérlő szavak

Valamely csatornán végbemenő be-, ill. kimeneti művelet vezérléséhez szükséges összes információt az egységvezérlő szó (EVSZ ill. UCW) tartalmazza. Az EVSZ részeit a 10.2. táblázatban láthatjuk. Az egységvezérlő szavak száma és formátuma különböző a szelektor- és multiplexcsatornák esetén. Ezeket az egységvezérlő szavakat a későbbiekben írjuk le.

10.2. táblázat

Az egységvezérlőszó részei

Forrás	Adat
B/K utasítás	Készülékcím
CAW	CCW cím, védelmi kulcs
CCW	Parancskód, jelzőbitek, számláló, adatcím
Csatorna	Csatornaállapot
B/K készülék	Készülékállapot

10.2.7. A programállapotszó

A programállapotszónak (PSSZ ill. PSW) csak egyes részei kerülnek felhasználásra a csatorna működtetéséhez. Ezek a csatornamaszkok a 0...2 közötti bitekben, a megszakítási kód a 16...31 közötti bitekben és a feltételkód a 34...35 közötti bitekben. A csatornamaszkok megakadályozhatják valamennyi B/K megszakítás felléptét; de ezek rendszerint az egyik csatornától engedélyezik míg a többiektől megakadályozzák a B/K megszakításokat.

Amikor a központi egység elfogad egy B/K megszakítást, a megszakítási kód megadja annak a csatornaegységnek a címét, amely a megszakítást okozta. A megszakítási kódot a régi programállapotszó tárolja. A feltételkódot a B/K utasítás végrehajtása során az Y2 és Y3 értékeinek megfelelően állítják be. A feltételkódnak a négy B/K utasításhoz való rendelését a 10.3. táblázat mutatja.

10.3. táblázat

A programállapotszó (PSW) feltételkódja be-, ill. kimeneti műveletekhez

Feltételkód	Rendeltetés
	B/K start
00	A B/K művelet eredményesen megkezdve
01	A CSW állapotrésze tárolva
10	A csatorna vagy alcsatorna foglalt
11	Nem működő egység
	B/K állj
00	A csatorna vagy alcsatorna nem működött
01	Kivétel feltétel, a CSW állapotrész tárolva
10	A művelet lezárult
11	Nem működő egység
	B/K teszt
00	Csatorna, egység és készülék rendelkezésre áll
01	CSW tárolva*
10	Csatorna vagy alcsatorna foglalt
11	Nem működő egység
	Csatorna teszt
00	Csatorna rendelkezésre áll
01	Felfüggesztett megszakítás a csatornában
10	Csatorna vagy alcsatorna foglalt
11	Nem működő egység

* Felszabadított megszakítás vagy kivétel feltétel a vizsgált B/K teszt alatt.

10.2.8.
Be-, ill. kimeneti (B/K) utasítások

A CPU négy különböző B/K utasítást használ a csatornával történő kommunikációra:

1. B/K start (START I/O),
2. B/K leállás (HALT I/O),
3. B/K teszt (TEST I/O) és
4. csatornateszt (TEST CHANNEL).

Ezeknek az utasításoknak SI formátumuk van, amint azt korábban már megismertük a 9.4. ábrából. A 8...15 bitet nem veszik figyelembe. A B mező (16...19 bit) egy regisztert jelöl ki. A B regiszter és a D mező (20...31 bit) tartalmának összege határozza meg a csatornát, az alcsatornát és a be-, ill. kimeneti készüléket. Ennek az összegnek a 21...31 bitjei képezik a 11-bites B/K címet, miközben a 0...20 biteket elhanyagolják. A 11-bites B/K címből a 21...23 bitek adják meg a csatorna címét, míg a 24...31 bit egy készülék címét adja és azonosítja a készüléket a csatornán, valamint multiplexcsatorna esetén ez határozza meg az alcsatornát.

1. A B/K START (I/O) utasítás írás, olvasás, visszafelé olvasás, vezérlés vagy érzékelés műveletet indít meg a megcímzett be-, ill. kimeneti készüléknél, a csatornánál vagy a részcsatornánál. A főtár 72. helyén levő csatornacímszó (CCSZ) tartalmazza a tárvédelmi kulcsot, amely a csatornához vagy az alcsatornához van most rendelve, és az első csatornaparancsszó címét. A csatornaparancsszó a következőket specifikálja:
— a műveletet,
— a felhasználandó főtártartományt és a
— művelet befejezésekor végrehajtandó tevékenységet.
Ennek az utasításnak a végrehajtásával állítódik be egyúttal a feltételkód a programállapotszóban, az állapot jelzésére a művelet végeztével.
2. A B/K leállásutasítás leállítja a pillanatnyi be-, ill. kimeneti műveletet a megcímzett alcsatornán vagy csatornán, és beállítja a programállapotszó feltételkódját az ok jelzésére.
3. A B/K tesztutasítás a programállapotszó feltételkódját állítja be a megcímzett csatorna, alcsatorna és készülék állapotának jelzésére és bizonyos körülmények között a csatornaállapot-szó tárolására.
4. A csatornateszt-utasítás a programállapotszó feltételkódját állítja be, ezzel jelezve a megcímzett csatorna állapotát; de ez a megcímzett csatorna állapotát nem érinti.
A programállapotszó feltételkódjának e négy B/K utasításhoz való hozzárendelését a 10.3. táblázat mutatja.

10.2.9.
Be-, ill. kimeneti (B/K) megszakítások

A B/K megszakítások a központi egység számára adnak lehetőséget arra, hogy az állapotát a be-, ill. kimeneti készülékekben vagy a csatornákban előforduló megszakítási viszonyok szerint változtassa. A megszakítási feltételt valamelyik készülék vagy az egyik csatorna kezdeményezheti. A készülék által kezdeményezett feltételek és azok típusai a következők:

1. Csatorna vég (végtípus),
2. Készülék vizsgálat (végtípus),
3. Különleges feltétel (végtípus),
4. Készülék vég (készülékvégtípus),
5. Vezérlőegység vég (készülékvégtípus),
6. Figyelem (készülékvégtípus),
7. Programellenőrzés (végtípus),
8. Programvezérelt megszakítás vagy PCI (végtípus).

Az utóbbi két feltétel a csatorna által kezdeményezett feltétel.

Minden egyes be-, ill. kimeneti művelet, amelyet a B/K start utasítás kezdeményez, a csatorna vég és a készülék vég feltételt generálja.

A *csatorna vég feltétel* előállítódik mindannyiszor, amikor a vezérlőegységnek többé nincs szüksége az interface-eszközökre a parancs befejezésére.

A *készülék vég feltétel* akkor generálódik, amikor a pillanatnyilag működő készülék eléri a normális mechanikai végpontját.

A *vezérlőegység vég feltétel* azért jön létre, hogy jelezze, a vezérlőegység most rendelkezésre áll. Ennek a feltételnek attól a vezérlőegységtől kell származnia, amelyet előzőleg B/K start- és B/K tesztutasítások címeztek meg, és akkor foglalt volt.

A *figyelemfeltételt* olyan készülékek állítják elő, mint pl. a konzolírógép és a display konzol, ahol kézi kérések történnek.

A *programvezérelt megszakítás feltétel* (PCI) akkor generálódik, ha PCI jelzőbit van a csatornaparancsszóban. Hacsak csatorna vége feltétel nem következik be, mielőtt a PCI megszakítás létrejönne, a PCI jelzőbit továbbterjed a következő csatornaparancsszavakon mindaddig, amíg a megszakítás be nem következik.

Amint az előzőekben bemutattuk, a megszakítási feltételeknek két típusa van.

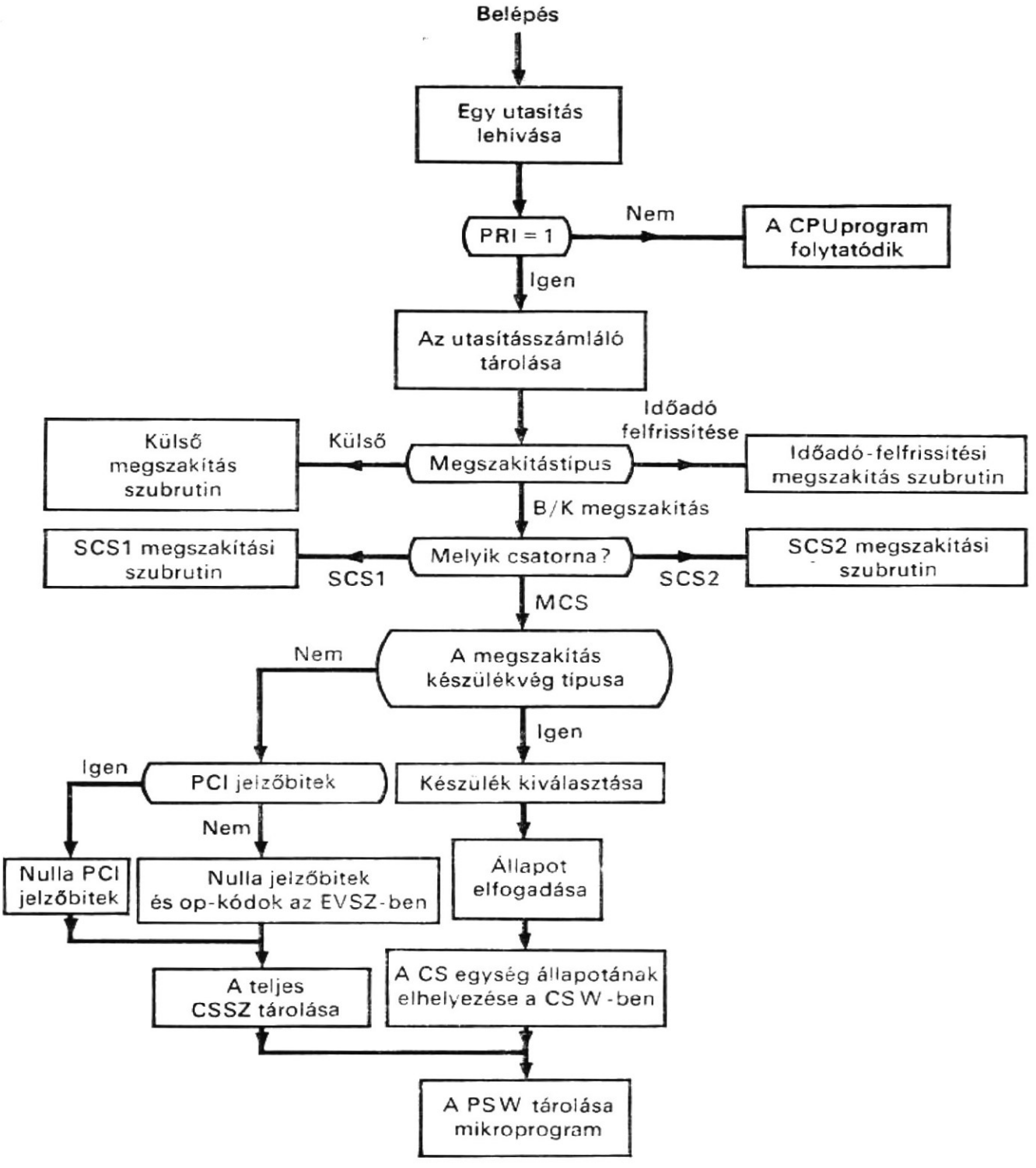
A **vég- (end) típusú megszakításban** egy teljes csatornaállapot-szót tárolnak a főtárban, míg az egység állapotát az egységvezérlő szóban, az alcsatornában tárolják. Tehát, a csatornaállapot-szó kialakításához szükséges információ az alcsatornában található, az egység pedig nem választható ki arra, hogy megkapjuk az állapotot.

A **készülékvég típusú megszakításban** a készülék állapotát a készüléknél helyezik le. A készüléket úgy választják ki, hogy ez az állapot elérhető és azután a csatornaállapot-szóban tárolható legyen. Röviden, a vég típusú megszakítás az állapotot alcsatornában teszi elérhetővé, míg a készülékvég típusú megszakításkor az állapot a készüléknél van elhelyezve.

Illusztrációképpen a 10.10. ábrán a multiplexcsatorna megszakítási mikroszubrutinjának, vagy más elnevezéssel mikroprogramjának erősen leegyszerűsített folyamatábráját láthatjuk. Az ábrán a PRI (**PR**ogram **I**nterrupt) program megszakításfeltételt képvisel, amelyet a külső megszakítások, B/K megszakítások és az időadó (timer) felfrissítésekor bekövetkező megszakítás állíthat be. Amint az ábrán látható, a PRI feltételt minden egyes utasítás lehívásakor megvizsgálják. Ha a PRI feltétel értéke 1, akkor az utasításszámlálót tárolják és meghatározzák a megszakítás okát (típusát). Ha az külső megszakítás vagy időadó felfrissítésekor bekövetkező megszakítás, a külső megszakítás mikroszubrutinnak, ill. az időadó felfrissítési megszakítás mikroszubrutinnak adják át a vezérlést. Ha viszont B/K megszakítás, meghatározzák a csatornát. Ha a megszakítás multiplexcsatornával kapcsolatos, akkor ezt követően meghatározzák, hogy a megszakítás készülékvég típusú-e, vagy vég típusú-e. Ha az vég típusú (a megszakítási puffer 0 byte-jának 5. bitje 0, amint később látni fogjuk), a teljes csatornaállapot-szót tárolják. Ha a megszakítás vég típusú, de nem PCI, 0-ra állítják az egységvezérlő szóban a PCI jelzőbiteket, és a műveleti kódokat. Ha ez végtípusból következik és PCI, akkor csak a PCI jelzőbitek kapnak 0 értéket. Ha az készülékvég típusú, kiválasztják a készüléket, a B/K teszt parancs felhasználásával, és az állapotát elfogadják oly módon, hogy később a csatornaparancsszó részeként kerül felhasználásra. Az előállított állapotot kiolvassák, majd törlik a készüléknél. A csatornaállapot-szót, amely csak egységállapotot és csatornaállapotot tartalmaz, a főtárban tárolják. A megszakítási mikroprogram hatására mindkét esetben programállapotszó tárolását végző mikroprogram veszi át a vezérlést.

10.3. Be-, ill. kimeneti (B/K) interface

A B/K interface azokat a síneket tartalmazza, amelyek kommunikációs kapcsolatot teremtenek a csatorna és a B/K vezérlőegységek között (és végső soron az egyes vezérlőegységekhez csatlakozó be-, ill. kimeneti készülékek között). A különböző típusú be-, ill. kimeneti készülékekhez — mint pl. írógépekhez, kártyaolvasókhoz, nyomtatókhoz, mágnesdobos és mágneslemezes tárakhoz — különböző vezérlőegységek szükségesek. A periféria vezérlőegység dekódolja a csatornából kapott parancsokat, értelmezi azokat a be-, ill. kimeneti készülék számára, továbbá jelsorozatot ad ki a parancsok végrehajtására. Minthogy a B/K vezérlőegységek különböző típusúak, a B/K vezérlőegység és a készülékek közötti illesztésnek is különböző



10.10. ábra
A multiplexcsatorna megszakítási folyamatábrája

típusúaknak kell lenniük. A számítógépcsaládok esetén mégis az vált gyakorlattá, hogy azonos illesztést használnak a csatorna és a hozzá kapcsolódó B/K vezérlőegységek között. Az ilyen, *standard B/K illesztésnek*, ill. standard interface-nek nevezett interface egységes módszert kínál a B/K vezérlőegységeknek a csatornához történő csatlakoztatásra, továbbá könnyebbé és gazdaságosabbá válik a számítógéprendszernek a különböző be-, ill. kimeneti készülékkel való összekapcsolása.

A következőkben az IBM System/360 családjának standard illesztését (standard interface-ét) írjuk le. Az illesztésnek közösek a formátumai és jelsorozatai valamennyi, hozzá kapcsolt vezérlőegység számára. A kiválasztás vezérlését végző jeleken kívül minden, a csatornához menő, ill. attól jövő átvitel egy „közös sínen" keresztül történik. Minden időegységben logikailag csak egy vezérlőegység tud a csatornához kap-

csolódni. A vezérlőegység kiválasztását a csatornával történő kommunikációra egy, az összes vezérlőegységen áthaladó vonal (ill. az azon megjelenő jel) vezérli. Ennek a (kezdeti) kiválasztó jelnek nevezett jel hatására az egyes vezérlőegységek válaszjelet hoznak létre, mely válasz a csatorna által szolgáltatott jelekre. A vezérlőegység mindaddig logikailag összekapcsolva marad az illesztéssel, amíg az a számára szükséges vagy a tőle küldendő információt továbbítja, vagy pedig, amíg a csatorna parancsot nem ad a szétválasztásra. A csatornához kapcsolódó B/K vezérlőegységek prioritása a csatornához való közelségük fokától függ.

10.3.1. A be-, ill. kimeneti interface multiplexrendszerben

Blaauw definíciója szerint [7], multirendszernek azt a rendszert nevezzük, amelyben kettő vagy több, egymással kézi beavatkozással vagy anélkül kommunikálni tudó központi feldolgozóegység van. A 10.11. ábrán látható multirendszer B/K interface lehetőségeit fogjuk a következőkben példaképpen megvizsgálni. Ebben a multirendszerben négy standard interface (B/K illesztés) van (az A, B, C és D), amelyek nagyszámú be-, ill. kimeneti készülékkel kapcsolják össze a két CPU-t.

Az **A illesztés** csatlakoztatja az A csatornát az X, Y, Z több készülékes vezérlőegységekhez és a T megosztva használt, több készülékes vezérlőegységhez.

A **D illesztés** kapcsolja a D csatornát az R több készülékes vezérlőegységhez, az integrált vezérlőegységekhez (azaz a vezérlőegység integrált része a két B/K készülékneknek) és a T megosztottan használt, több készülékes vezérlőegységhez. Mivel a megosztottan használt (más szóval közös) többkészülékes vezérlőegységek az A és D illesztésekhez csatlakoznak, a két, hozzájuk kapcsolt be-, ill. kimeneti készülék az 1. központi egységet az A csatornán, a 2. központi egységet pedig a D csatornán keresztül tudja elérni.

A **B illesztés** a B csatornát a csatorna—csatornaadapterhez és a W több készülékes vezérlőegységhez kapcsolja.

A **C illesztés** a C csatornát csatlakoztatja a csatorna—csatornaadapterhez és a V több készülékes vezérlőegységhez.

A csatorna—csatornaadapter két csatorna B/K illesztéseinek összekapcsolását teszi lehetővé, és ezáltal az egyik csatorna a másik számára vezérlőegységként jelenik meg; tehát, a CPU-k össze vannak egymással kapcsolva. Továbbá, négy be-, ill. kimeneti készülék csatlakozik egy megosztottan használt kapcsolóhoz, amely viszont az R, V, W és X több készülékes vezérlőegységekhez kapcsolódik. Ez a szervezés a négy be-, ill. kimeneti készülék megosztott használatát teszi lehetővé a négy vezérlőegység részére, és elérését a két CPU által, a négy illesztés bármelyikén keresztül.

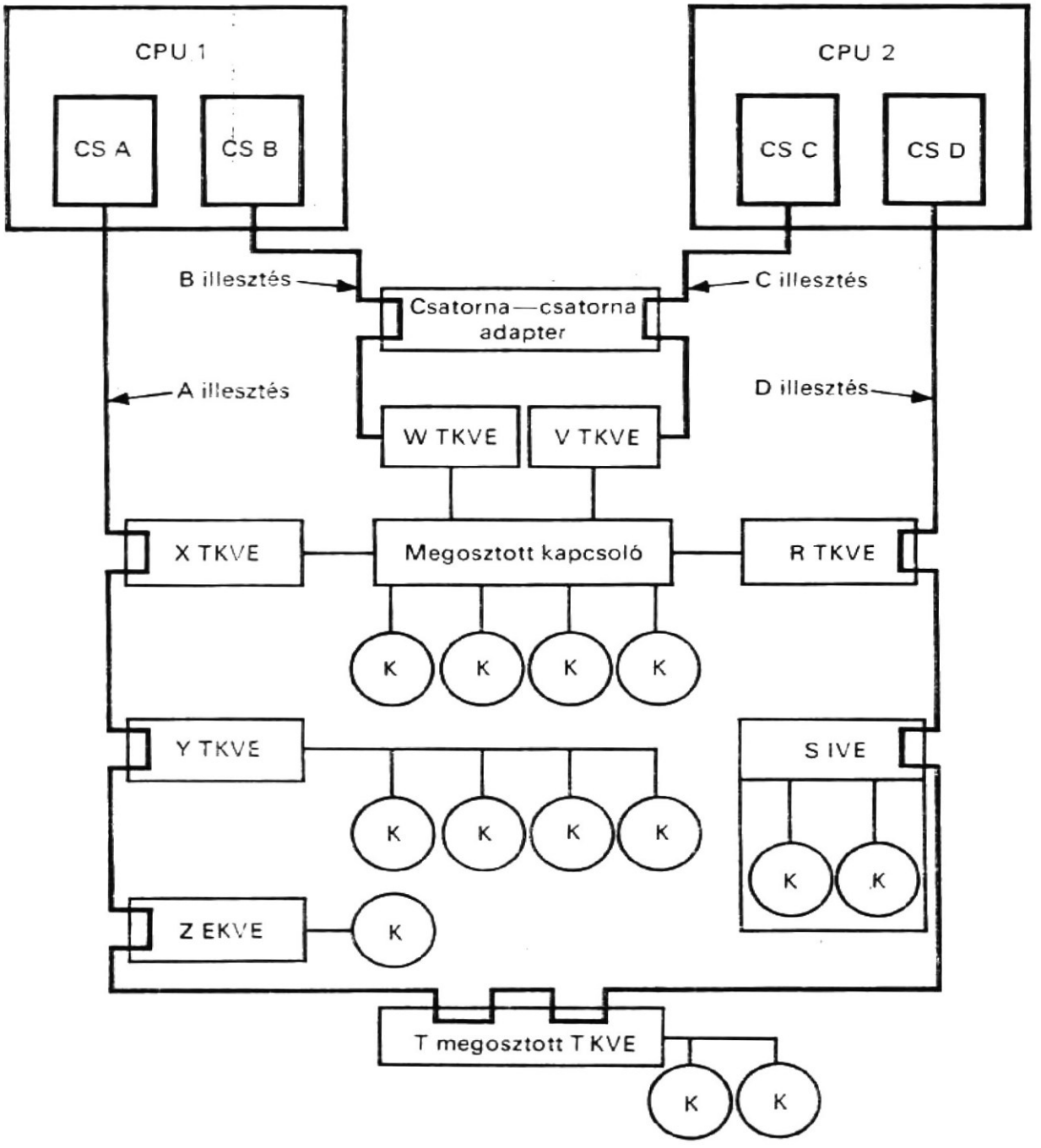
10.3.2. Illesztősín

A standard B/K illesztés sínrendszere 34 vonalból áll; ezeket hét csoportba lehet osztani:

— bemeneti sín,
— kimeneti sín,
— bemeneti összekötésjelzések (inbound tag),

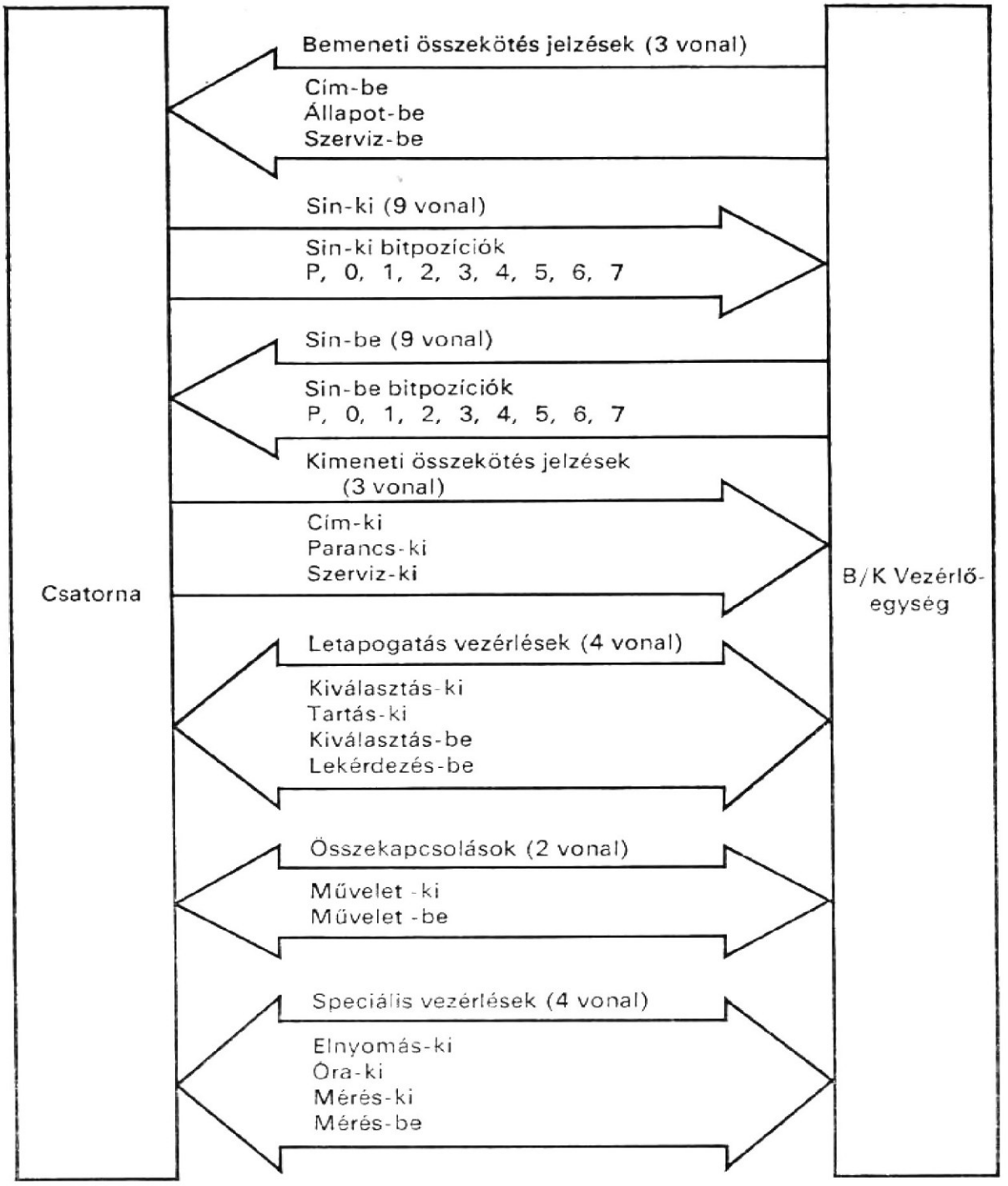
— kimeneti összekötésjelzések (outbound tag) letapogatás vezérlések,
— letapogatás vezérlések
— összekapcsolások (interlock) és
— speciális vezérlések.

Ezeket a sineket a 10.12. ábrán láthatjuk, és a következő terminálutasításokkal deklaráljuk őket. Mivel ez a leírás standard interface-re vonatkozik, ezért a standard kifejezések nem változtathatók meg. Az utasítások tehát az eredeti angol kifejezések rövidítései.



CPU: Központi feldolgozóegység (KFE)
EKVE: Egyedi készülék vezérlőegysége
TKVE: Több készülék vezérlőegysége
Cs: Csatorna
K: B/K készülék
IVE: Integrált vezérlőegység

10.11. ábra
Multiplex rendszer B/K illesztései



10.12. ábra
Az IBM System/360 szabványos B/K illesztésének 34 vonala

```
Comment, a 34 illesztési vonal deklarációja                    (10.2)
Comment, két sín
Terminal,  BUS—IN(P, 0—7),      $8 adatvonal és 1 paritásvonal
           BUS—OUT(P, 0—7),     $8 adatvonal és 1 paritásvonal

Comment, három bemeneti összekötés és három
         kimeneti összekötés jelzés vonal
Terminal,  ADR—IN               $cím be vonal
           STA—IN               $állapot be vonal
           SRV—IN               $szerviz be vonal
           ADR—OUT              $cím ki vonal
           CMD—OUT              $parancs ki vonal
           SRV—OUT              $szervisz ki vonal
```

```
Comment, négy letapogatás vezérlés vonal
Terminal,  SEL—IN        $kiválasztás be vonal
           REQ—IN        $lekérdezés be vonal
           SEL—OUT       $kiválasztás ki vonal
           HLD—OUT       $tartás ki vonal

Comment, két összekapcsolás vonal
Terminal,  OP—IN         $művelet (op) be vonal
           OP—OUT        $művelet ki vonal

Comment, négy speciális vezérlővonal
Terminal,  MTR—IN        $mérés be vonal
           MTR—OUT       $mérés ki vonal
           SUP—OUT       $elnyomás ki vonal
           CLK—OUT       $óra ki vonal
```

A BUS—OUT vonalakat a csatornától a vezérlőegységhez menő adatok, címek, parancsok vagy vezérlőutasítások átvitelére használják; a BUS—IN vonalakat pedig a vezérlőegységtől a csatorna felé áramló adatok, készülékazonosítás, állapotinformáció és adatérzékelés átvitelére. A jelzésvonalakat a síneken levő információ összekapcsolására és jelzésére, valamint speciális sorozatok céljára használják. A letapogatásvezérlés a csatlakoztatott be-, ill. kimeneti készülékek letapogatására vagy kiválasztására szolgál. A speciális vezérlési vonalakat a különböző csatlakoztatott egységekben elhelyezett érzékelőkhöz használják.

10.3.3. Vezérlések ütemezése

Az illesztési vonalak kapcsolják össze, a csatornához rendelt vezérlőegységet a csatornával. Egy vagy több ilyen vonal jelei jelsorozatot alkotnak, amelyek a vezérlési információ továbbítását végzik. Ilyen sorozatok kialakítását láthatjuk a 10.13. ábrán, értelmezésüket az alábbiakban végezzük el.

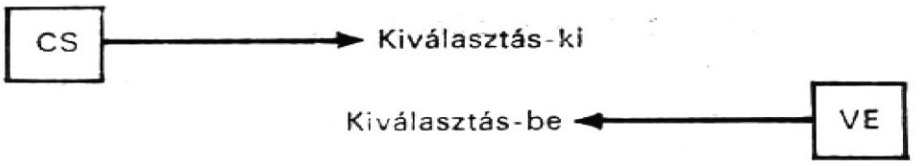
Valamelyik vezérlőegység kiválasztása a SEL—OUT és a SEL—IN vezetékekkel történik. Ezek hurkot képeznek oly módon, hogy a SEL—OUT vezeték összeköti a csatornát a vezérlőegységekkel, sorban egyiket a másikkal, majd a SEL—IN vezeték a kört zárva a legutolsó vezérlőegységből kiindulva a csatornához kapcsolódik. Amikor a csatorna felemeli a SEL—OUT vezetéken levő jelet, ez a jel egymás után, a csatornához való távolság függvényében lekérdezi az egyes vezérlőegységeket; ezt a folyamatot *lekérdezésnek* (polling) nevezik. A SEL—OUT vezetéken levő jel felemelése a SEL—IN vezetéken olyan jelet állít elő, amely arról értesíti a csatornát, hogy az illesztésen egyetlen vezérlőegység sem kért kiszolgálást (10.13*a* ábra).

Az ADR—OUT vezetéket arra használják, hogy az összes vezérlőegység számára eljuttassa és dekódolja a BUS—OUT vonalon levő be-, ill. kimeneti készülék címet; ha valamelyik vezérlőegység felismeri a címet, az OP—IN vezetéken levő jel felemelésével válaszolnia kell, ezzel jelezve, hogy a készülék kiválasztása megtörtént (10.13*b* ábra).

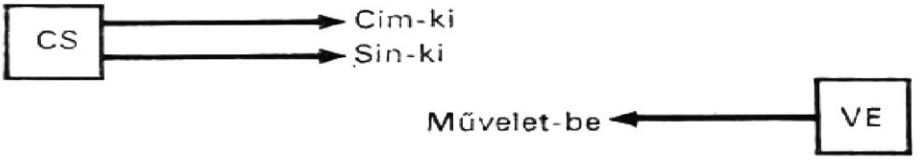
Valamelyik vezérlőegység akkor emeli fel a REQ—IN vezetéken levő jelet, amikor jelezni akarja a csatornának, hogy kiszolgálást kér. A csatorna a SEL—OUT vezetéken levő jel felemelésével válaszol (10.13c ábra).

Az ADR—IN vezeték segítségével a vezérlőegység jelzi a csatornának, hogy a pillanatnyilag kiválasztott be-, ill. kimeneti készülék címe a BUS—IN vezetéken van, a

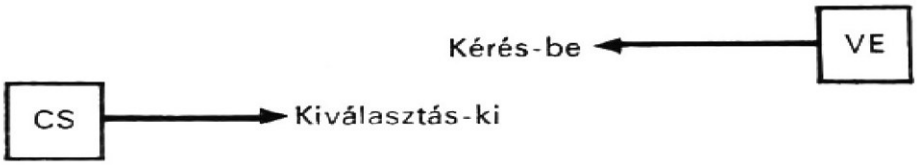
a) Lekérdezés



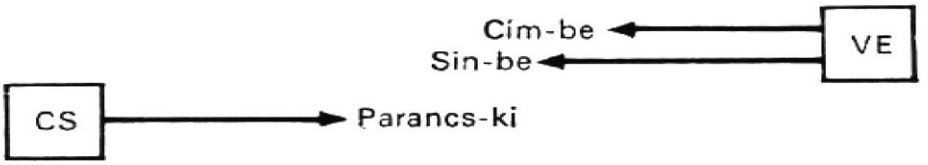
b) Készülékkiválasztás



c) Szervizkérés

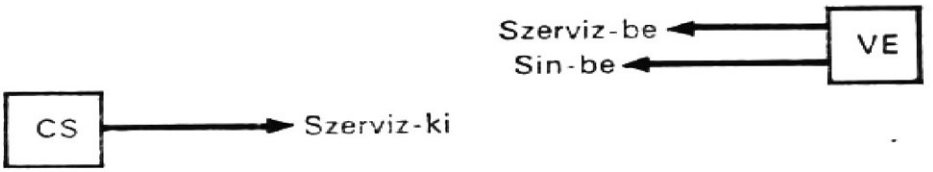


d) Címelfogadás

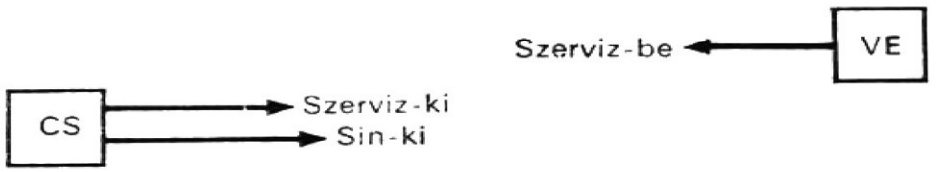


e) Állapotelfogadás

f) Adatelfogadás

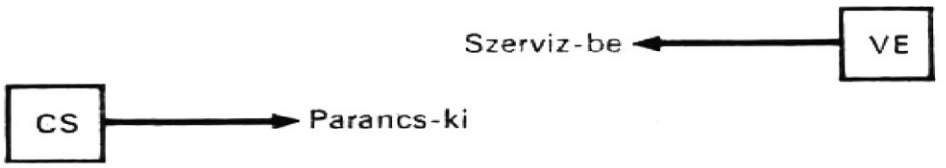


g) Adatkész

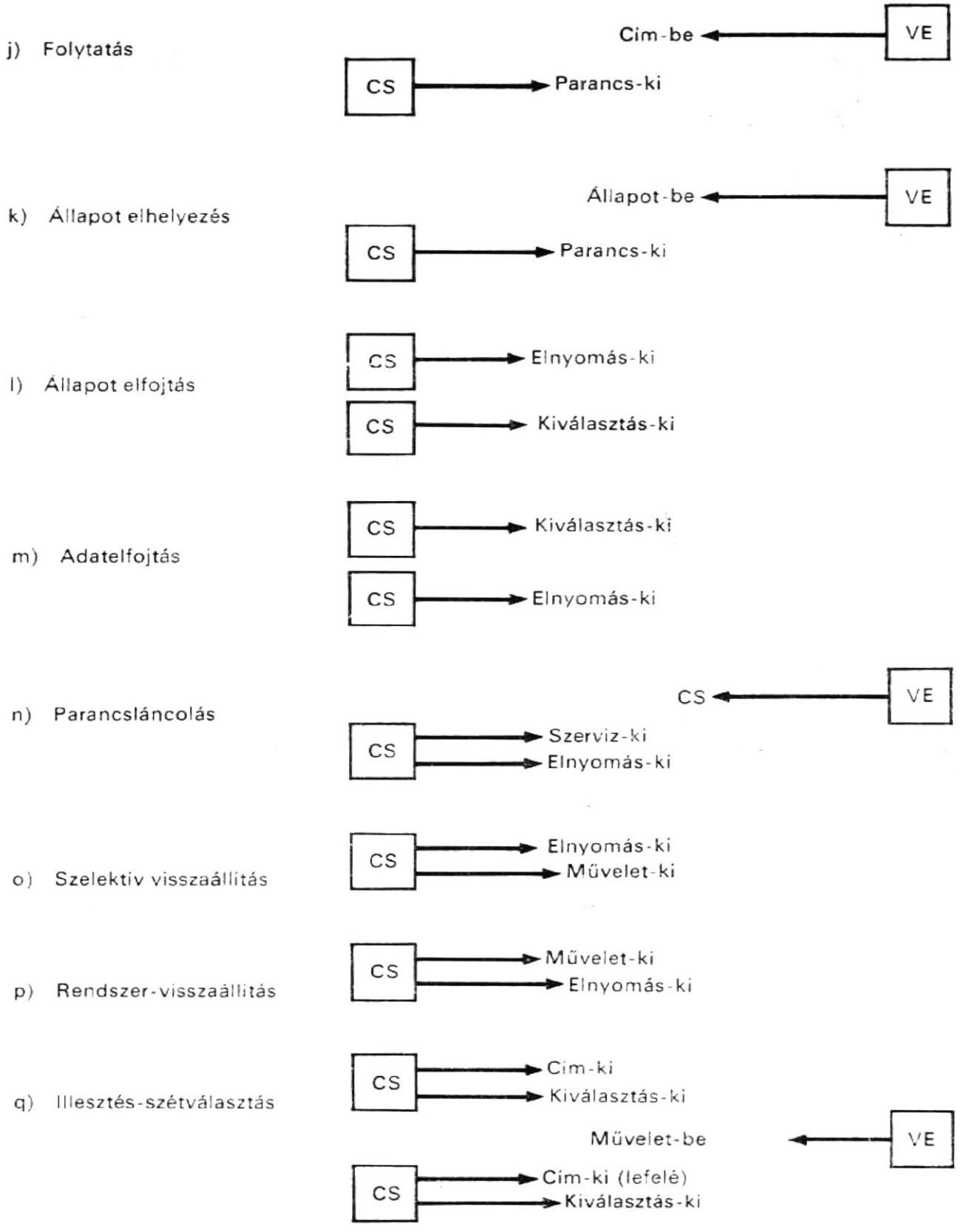


h) Parancskész

i) Stop



10.13/a. ábra



10.13. ábra
Vezérlések ütemezése

csatorna erre a CMD—OUT vezetéken válaszol, jelezve, hogy a címet elfogadta (10.13*d* ábra).

Az STA—IN vezetéket arra használja a kiválasztott vezérlőegység, hogy jelzést küldjön a csatornának: állapotinformációt helyezett el a BUS—IN vonalon: a csatorna elfogadta az állapotinformációt (10.13e ábra).

Az SRV—IN vezetéket a vezérlőegység arra használja, hogy a csatornának jelezzen, amikor a kiválasztott be-, ill. kimeneti készülék kész egy byte-nyi információ továbbítására vagy fogadására. Ha a csatorna olvasás, visszafelé olvasás vagy érzékelés művelet alatt az SRV—OUT vezetéken válaszként jelzést ad az SRV—IN vezetéknek, ez azt jelenti, hogy elfogadta a BUS—IN vezetéken elhelyezett információt (10.13*f* ábra). Ha ez írás vagy vezérlési művelet közben történik, azt jelenti, hogy a BUS—OUT vezetéken elhelyezkedő kívánt információ készen áll arra, hogy a vezérlőegység fogadja (10.13*g* ábra).

A CMD—OUT vezeték alapvető funkciója annak jelzése a vezérlőegység számára, hogy a BUS—OUT vezetéken levő információ parancs (10.13*h* ábra).

Ha a CMD—OUT vezetéken levő jelet az SRV vezetéken levő jelre adott válaszként küldik, ez stoppot jelez, vagyis a csatorna éppen befejezi a pillanatnyi műveletet (10.13*i* ábra).

Ha a CMD—OUT vezetéken levő jelet az ADDR—IN vezetékre adott válaszként küldik, ez a vezérlőegységnek azt jelzi, hogy folytassa a működését (10.13*j* ábra).

Ha a CMD—OUT az STA—IN-re adott választ jelent, a csatorna arra kéri a vezérlőegységet, helyezze el az állapotinformációt a vezérlőegységben, vagy a be-, ill. kimeneti készülékben (10.13*k* ábra).

Az SUP—OUT funkciói a következők:

— állapotelnyomás,
— adatátvitel-elnyomás,
— parancsláncolás,
— szelektív visszaállítás (reset) (10.13*o* ábra).

Amikor SEL—OUT jel jön létre olyan vezérlőegységen, amely lerakott állapotinformációval rendelkezik, ez a vezérlőegység nem hoz létre illesztést az állapotinformáció közlésére, ha aktív SUP—OUT=1, vagy a vezetéken levő jel fel van emelve, tehát az állapotinformáció elnyomódik (10.13*l* ábra). Ha a csatorna az SUP—OUT vezetéket a SEL—OUT vezetéken levő jel leesése előtt emeli fel, az OP—IN és a REQ—IN blokkolva van a vezérlőegység felől, és az adatátvitel elfojtódik (10.13*m* ábra).

Ha a csatorna, a vezérlőegységtől jövő STA—IN-re adott válaszként az SUP—OUT és SRV—OUT jeleket küldi, a kiválasztott vezérlőegység és a be-, ill. kimeneti készülék tartásállapotba kerül, és a csatorna által küldött következő parancs kerül erre a vezérlőegységre, és a be-, ill. kimeneti készülékre (10.13*n* ábra).

Az OP—OUT valamennyi, az interface-en (illesztésen) levő vezérlőegységnek lehetővé teszi, hogy a csatornával kommunikáljon. Ha az OP—OUT leesett és az SUP—OUT felemelkedett, miközben a vezérlőegység az illesztésen működik, a vezérlőegységet viasszaállítják (10.13*o* ábra).

Ha mind az OP—OUT, mind az SUP—OUT leesett, az összes — az illesztésen levő — vezérlőegységet visszaállítják (10.13*p* ábra).

Ha az ADR—OUT felemelkedett, és az SUP—OUT leesett, bármelyik jelsorozat befejezése előtt, a vezérlőegység felismeri az illesztésszétválasztás jelet, és az OP—IN vezetéken levő jel leesésével fog válaszolni, míg a csatorna válasza az ADR—OUT vezetéken levő jel leesése lesz (10.13*q* ábra).

A csatorna a HLD—OUT vezetéken ad jelet, párhuzamosan az összes vezérlőegységnek, hogy engedélyezzék a csatornának a SEL—OUT vezeték hatásának érvénytelenítését a kiválasztott vezérlőegységen. Ennek eredményeképpen a SEL—OUT vezeték nem gyakorol hatást a vezérlőegységre. A CLK—OUT azt jelzi a vezérlőegységnek, hogy a CPU nincs sem leállás, sem várakozás állapotban. Az MTR—OUT vezeték a vezérlőegységnél levő működési időmérőegységet aktivizálja az idő jegyzésére. Az MTR—IN vezeték azt jelzi a csatornának, hogy a vezérlőegység méri az időt.

10.3.4.
Cím-, parancs-, állapot- és érzékelésbyte-ok

A BUS—OUT vezetékeken levő információbyte lehet cím vagy parancs, míg a BUS—IN vezetékeken cím, állapotinformáció vagy érzékelésinformáció. Ennek megfelelően címbyte-nak, parancsbyte-nak, állapotbyte-nak és érzékelésbyte-nak nevezik őket.

Amikor egy vezérlőegység működésbe lép, mindegyik, hozzákapcsolódó be-, ill. kimeneti készülék kap egy egyedi 8-bites készülékcímet. Ezt a *címbyte-ot* használják föl az illesztésen keresztül a csatlakoztatott be-, ill. kimeneti készülékek direkt címzésére. Továbbá, a vezérlőegységhez címbyte-ok halmazát küldik el, és ha felismeri saját címét, amelyek készen vannak, vagy amelyek — bár nincsenek készen —, de készenlétbe helyezhetők egyszerű kézi beavatkozással, válaszolnia kell.

Az illesztésen keresztül végrehajtandó be-, ill. kimeneti műveletet a csatorna által a be-, ill. kimeneti készülékhez küldött *parancsbyte* határozza meg, a csatorna által kezdeményezett kezdeti kiválasztási folyamat alatt. A parancsbyte felépítését a 10.4. táblázat mutatja. Amint ott látható, a kisebb helyiértékű bitpozíciók határozzák

10.4. táblázat

Az illesztésen megjelenő parancsbyte felépítése

Parancs	Bitpozíció								
	P	0	1	2	3	4	5	6	7
Test I/O	1	0	0	0	0	0	0	0	0
Érzékelés	P	M	M	M	M	0	1	0	0
Olvasás hátra	P	M	M	M	M	1	1	0	0
Írás	P	M	M	M	M	M	M	0	1
Olvasás	P	M	M	M	M	M	M	1	0
Vezérlés	P	M	M	M	M	M	M	1	1
Bázisérzékelés	0	0	0	0	0	0	0	0	0
Bázisolvasás	0	0	0	0	0	0	0	1	0
No-op vezérlés	1	0	0	0	0	0	0	1	1

M a módosítóbit; P a paritásbit.

meg a műveletek típusát. A nagyobb helyértékű bitpozíciók módosítóbitek, amelyeket a vezérlőegység és a be-, ill. kimeneti készülék használhat. A módosítóbitek pontos értékei az adott be-, ill. kimeneti készüléktől függnek. Figyeljük meg, hogy a 10.4. táblázatban a bázisérzékelés, bázisolvasás és no-op vezérlés olyan érzékelési, olvasási, ill. vezérlési parancs, amelynek módosítóbitje 0.

Az olvasásparancs adatátvitel végrehajtását kezdeményezi a vezérlőegységtől a csatornához. Az adatokat pl. a működésben levő adott be-, ill. kimeneti készülék rekord forrásából nyerik. Az *olvasás hátra* parancs ugyanolyan módon kezdeményez működést, mint az *olvasásparancs*, azzal az eltéréssel, hogy az adatbyte-okat a főtárba a csatorna az olvasási parancshoz képest fordított sorrendben viszi be. Az olvasási parancs a csatornától a vezérlőegységhez történő adatátvitelt kezdeményezi. A vezérlőegység dekódolja a módosítóbiteket, és végrehajtja a kívánt műveleteket.

Az érzékelésparancs pontosan úgy megy végbe, mint az olvasási parancs, azzal az eltéréssel, hogy az adat egy *érzékelésbyte*.

A be-, ill. kimeneti teszt parancs az *állapotbyte-ot* a megcímzett be-, ill. kimeneti készüléktől kapja.

A bázisolvasás parancs tulajdonképpen nullát módosító bittel ellátott olvasási parancs. Ezt olvasási parancsnak is használják a kezdeti-program betöltésekor (Initial Program Loading: IPL). A bázisérzékelési parancs érzékelési műveletet kezdeményez az összes be-, ill. kimeneti készüléken. A no-op (nincs működés) vezérlésparancs semmilyen műveletet nem hajt végre a be-, ill. kimeneti készüléken, kivéve, hogy kielégíti az előzetesen jelzett láncolási műveleteket és megengedi bizonyos be-, ill. kimeneti készüléknek, hogy ellenőrzési feltételekre várjanak a csatorna felszabadítása előtt.

Az állapotbyte felépítését a 10.5. táblázat mutatja. Az állapotbyte a következő hat állapot valamelyikébe kerül átvitelre a csatornához:

1. A kezdeti kiválasztási sorozat alatt.
2. A csatorna vég állapot közlésére az adatátvitel befejezésekor.
3. A készülékvég jel és mindenfajta kapcsolódó feltétel közlésére a csatorna számára.
4. A vezérlőegység vége vagy készülékvég állapot közlésére, ami azt jelzi, hogy a vezérlőegység vagy a készülék szabad.
5. Bármilyen, előzetesen meghatározott állapot közlésére, ha engedélyezett ennek elvégzése.
6. Bármilyen külsőleg kezdeményezett állapot közlésére a csatorna számára.

Az érzékelési művelet alatti adatátvitel a legutolsó művelet alatt érzékelt szokatlan feltételről szóló információt ad, amely a be, ill. kimeneti készülék aktuális állapotát jelzi. Rendszerint az összes érzékelési információt az első két byte-ben helyezik el. Az ezeket követő bármelyik bitpozíció, amelyeket programozási információra használnak, diagnosztikai információt tartalmaz, és ez annyi byte-ra terjedhet ki, amennyi éppen szükséges. A legtöbb be-, ill. kimeneti készülék esetén az első érzékelési byte első hat bitje közös az összes be-, ill. kimeneti készülék számára. Felépítésüket a 10.6. táblázat mutatja.

10.5. táblázat

A státuszbyte felépítése

Bitpozíció	Rendeltetés
P	Paritás
0	Figyelem
1	Állapotmódosító
2	Vezérlőegységvég
3	Foglalt
4	Csatornavég
5	Készülékvég
6	Készülékvizsgálat
7	Különleges feltétel

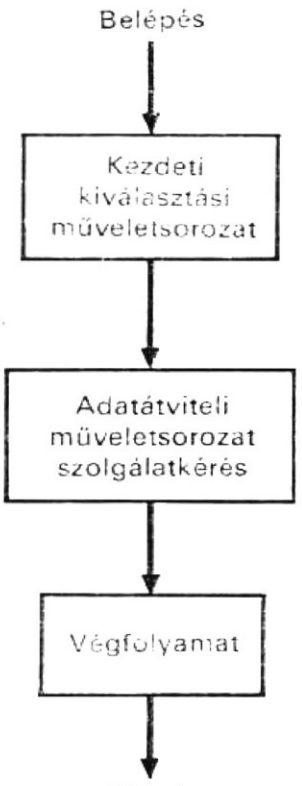
10.6. táblázat

Az első érzékelésbyte felépítése

Bitpozíció	Rendeltetés
0	Parancsvisszautasítás
1	Beavatkozás szükséges
2	Sin-ki ellenőrzés
3	Berendezésellenőrzés
4	Adatellenőrzés
5	Túlfutás

10.3.5. Interface-sorozatok

A be-, ill. kimeneti illesztésen az információátvitelt a rajta keresztül haladó különféle jelsorozatok hajtják végre. Példának a következőkben a multiplexcsatorna egyik alcsatornáján végbemenő byte-üzemmódú műveletet választottuk. A művelet folyamatábráját a 10.14. ábra mutatja. Amint itt látható, a multiplexcsatorna kezdeti kiválasztási folyamattal (műveletsorozattal) indul, amelyben kiválasztja a be-, ill. kimeneti készüléket. Miután ez megtörtént, átviszi az adatot az adatátviteli műveletsorozat aktivizálásával. Az adatátvitel befejezésekor lezárási sorozatot hajt végre



10.14. ábra
Byte-üzemmódú működés a multiplexcsatorna egyik alcsatornájában

(végfolyamat). Ha egynél több be-, ill. kimeneti műveletre van szükség, a multiplexcsatorna sorrendben hajtja végre ezeket a műveleteket, az aktív vezérlőegységek prioritásának megfelelően. Az illesztési műveletsorozatok működésének illusztrálásaként az alábbiakban példán bemutatjuk a kezdeti kiválasztási folyamatot, a byte-üzemmódú adatátviteli folyamatot és a végfolyamatot (lezárási sorozatot).

A *kezdeti kiválasztási* folyamat lehetőséget ad a csatornának, hogy kiválasszon egy be-, ill. kimeneti készüléket, és specifikálja a készülék működését. Akkor kezdődik, amikor a csatorna egy címbyte-ot továbbít az illesztéshez. Az interface-hez csatlakozó összes vezérlőegység kísérletet tesz a cím dekódolására, de csak egyetlen vezérlőegység tudja értelmezni a címbyte-ot. A kiválasztott vezérlőegység válaszaként vagy az STA—IN vezetéken levő jel emelkedik fel, jelezve, hogy az egység foglalt, vagy az OP—IN vezetéken levő jel emelkedik fel, jelezve, hogy a készülék be fogja fejezni a kezdeti kiválasztási folyamatot. Ha egyetlen vezérlőegység sem dekódolja a címbyte-ot, a legkisebb prioritású vezérlőegység visszaküldi a jelet a SEL—IN vezetéken a csatornának; ennek hatására az ADR—OUT vezetéken levő jel leesik és lezárja a kezdeti kiválasztási folyamatot. Amikor a csatorna megkapja a jelet az OP—IN vezetéken, az ADR—OUT vezetéken levő jel leesésével válaszol. A kiválasztott vezérlőegység ezután a csatornának továbbítja a címbyte-ot. Ekkor a csatorna ezt a címet összehasonlítja azzal, amelyet korábban a BUS—OUT vezetéken jelyezett el. Ha a cím egyezik, a csatorna egy parancsbyte-ot küld a vezérlőegységhez, amely állapotbyte küldésével válaszol. Ha a be-, ill. kimeneti készülék rendelkezésre áll, az állapotbyte tartalma nulla. Ha a csatorna elfogadja ezt az állapotbyte-ot, lezárja a kezdeti kiválasztási folyamatot az SRV—OUT vezetéken levő jel felemelkedésével.

Tételezzük föl, hogy az ISS regiszter adja ki az indító jelet a kezdeti kiválasztási műveletsorozathoz, és hogy a kiválasztás sikeres. Az illesztésen megjelenő jelsorozatot az alábbiakban írjuk le.

Comment, a kezdeti kiválasztási folyamat leírása (10.3)

```
/ISS/      ADR—OUT←1,
           BUS—OUT←„ki címbyte",
/ΔISS/     SEL—OUT←1,   $Δkésleltetés
```

```
Comment, a kiválasztott vezérlőegység válaszol
/SEL—OUT/     OP—IN←1,
/OP—IN/       ADR—OUT←0,

Comment, a kiválasztott vezérlőegység átviszi a címbyte-ot
/ADR—OUT'/    BUS—IN←„be címbyte",
              ADR—IN←1,

Comment, a csatorna összehasonlítja a címeket és ha megegyeznek, átviszi a
         parancsbyte-ot
/ADR—IN/      IF (be cím=ki cím) THEN
              (BUS—OUT←„parancsbyte," CMD—OUT←1, SEL—OUT←0),

Comment, a vezérlőegység cím be jele leesik
/CMD—OUT/     ADR—IN←0,

Comment, a csatorna parancs ki jele leesik
/ADR—IN'/     CMD—OUT←0,

Comment, a vezérlőegység átviszi az állapotbyte-ot
/MD—OUT'/     BUS—IN←„állapotbyte", STA—IN←1,

Comment, a csatorna és a vezérlőegység befejezi a kezdeti kiválasztási folyamatot
/STA—IN/      SVR—OUT←1,
/SRV—OUT/     STA—IN←0, OP—IN←0,
/STA—IN'/     SRV—OUT←0,
```

A byte-üzemmódú adatátviteli sorozat a következőképpen mehet végbe. A vezérlőegység a REQ—IN vezetéken levő jelet felemeli a lekérdezés vezérlésére. Amikor a csatornától jövő SEL—OUT vezeték letapogatja a vezérlőegységet, a vezérlőegység egy címbyte-ot küld. A csatorna fogadja a címet és utasítja a vezérlőegységet a folytatásra; eközben a csatorna kezdeményezi és végrehajtja a dump (töltő) rutint. Ekkor a vezérlőegység egy adatbyte-ot visz át; a csatorna fogadja az adatot és a undump (visszatöltő) rutint indítja és hajtja végre. A vezérlőegység ezzel befejezi az egy byte átvitelét.

Tételezzük fel, hogy DUMP (TÖLT) és UNDUMP (VISSZATÖLT) a töltő és visszatöltő rutinok aktivizálását végző vezérlőjelek és a DTS (ADAT) regiszter kezdeményezi az adatátviteli műveletsorozatot. Az alábbiakban leírjuk az adatátvitelhez szükséges és az illesztésen megjelenő jelsorozatot.

```
Comment, byte-üzemmódú adatátviteli jelsorozat multiplexcsatornán          (10.4)
Comment, a vezérlőegység kérési ciklust kér a csatornától
/DTS/         REQ—IN←1,

Comment, a csatorna letapogatja a vezérlőegységeket
/REQ—IN/      SEL—OUT←1,
Comment, a vezérlőegység egy címbyte-ot küld
/SEL—OUT/     BUS—IN←„címbyte",
              ADR—IN←1, OP—IN←1, REQ—IN←0,

Comment, a csatorna ezt fogadja és utasítást ad a folytatásra
/ADR—IN/      CMD—OUT←1, SEL—OUT←0, DUMP←1,

Comment, a vezérlőegység leejti a cím be jelet
/CMD—OUT/     ADR—IN←0,
```

```
Comment, a csatorna leejti a parancs ki jelet
/ADR—IN'/       CMD—OUT←0,

Comment, a vezérlőegység egy adatbyte-ot küld
/CMD—OUT'/      BUS—IN←„adatbyte",
                SRV—IN←1,

Comment, a csatorna nyugtáz és visszatöltést hajt végre
/SRV—IN/        SRV—OUT←1, UNDUMP←1,
                memory←„adat",

Comment, a csatorna és a vezérlőegység befejezi az átvitelt
/SRV—OUT/       SRV—IN←0, OP—IN←0,
/SRV—IN/        SRV—OUT←0,
```

Amikor bármelyik be-, ill. kimeneti művelet (kivéve a be-, ill. kimeneti teszt és no-op műveletet) előírt módon befejeződik, a vezérlőegység egy másik állapotbyte-ot küld a csatornához, kérve az átviteli folyamat lezárását. A csatorna jelt ad az SRV—OUT vezetéken annak jelzésére, hogy elfogadja az állapotbyte-ot és visszaállítja a működést. (A csatorna a CMD—OUT vezetéken is adhat jelet, ezáltal utasítva a vezérlőegységet, hogy helyezze el az állapotbyte-ot.) A következőkben leírjuk az illesztésen megjelenő jelsorozatot az END (VÉGE) regiszter által kezdeményezett végfolyamathoz (átvitelek lezárásához).

```
Comment, a végfolyamat leírása                                    (10.5)
Comment, a vezérlőegység szolgálatot kér a csatornától
/END/           REQ—IN←1,

Comment, a csatorna letapogatja a vezérlőegységeket
/REQ—IN/        SEL—OUT←1,

Comment, a vezérlőegység egy címbyte-ot küld
/SEL—OUT/       REQ—IN←0, OP—IN←1, ADR—IN←1,
                BUS—IN←„címbyte",

Comment, a csatorna fogadja és utasítást ad a folytatásra
/ADR—IN/        CMD—OUT←1, SEL—OUT←0,

Comment, a vezérlőegység leejti a címbe jelet
/CMD—OUT/       ADR—IN←0,

Comment, a csatorna leejti a parancs ki jelet
/ADR—IN'/       CMD—IN←0,

Comment, a vezérlőegység átviszi az állapotbyte-ot
/CMD—OUT'/      BUS—IN←„állapotbyte",
                STA—IN←1,

Comment, a csatorna nyugtázza az állapotbyte-ot
/STA—IN/        SRV—OUT←1,

Comment, a csatorna és a vezérlőegység lezárja az átvitelt
/SRV—OUT/       STA—IN←0, OP—IN←0,
/STA—IN'/       SRV—OUT←0,
```

10.4. Szelektorcsatorna

A szelektorcsatorna teszi lehetővé az IBM S/360 40-es modellje számára a nagy sebességű be-, ill. kimeneti készülékek használatát. A nagy adatátviteli sebességet úgy érik el, hogy külön regisztereket használnak a CPU regiszterei helyett a csatornafunkciók ellátására, és huzalozott logikai vezérléssel helyettesítik a hosszadalmas mikroprogramozott rutinokat. Összesen két szelektorcsatornát lehet a 40-es modellhez kapcsolni.

10.4.1. Egységvezérlő szó

Az egységvezérlő szót, EVSZ-t (**U**nit **C**ontrol **W**ord: UCW) a helyi tárban tárolják. Minden szelektorcsatornához csak egy EVSZ tartozik. Az 1. SZCS (**SZ**elektor-**CS**atorna) EVSZ-e a 32...37 helyet foglalja el, míg a 2. SZCS-hez tartozó EVSZ a 48...53 helyet. A szelektorcsatornák egységvezérlő szavának formátumát a 10.7. táblázat mutatja. Hat félszó van, ezek közül az EVSZ0 és EVSZ1 az A, ill. a D regiszter

10.7. táblázat

Szelektorcsatorna egységvezérlőszavának formátuma

Hely	Leírás
EVSZ0(0—15)	Dump (töltési) terület az A regiszter számára
EVSZ1(0—15)	Kitöltési terület a D regiszter számára
EVSZ2(0—15)	Következő CCW cím
EVSZ3(0—15)	A CCW cím újratöltése íráskor
EVSZ4(0—15)	Megszakítási puffer
EVSZ5(0—15)	Munkamező

10.8. táblázat

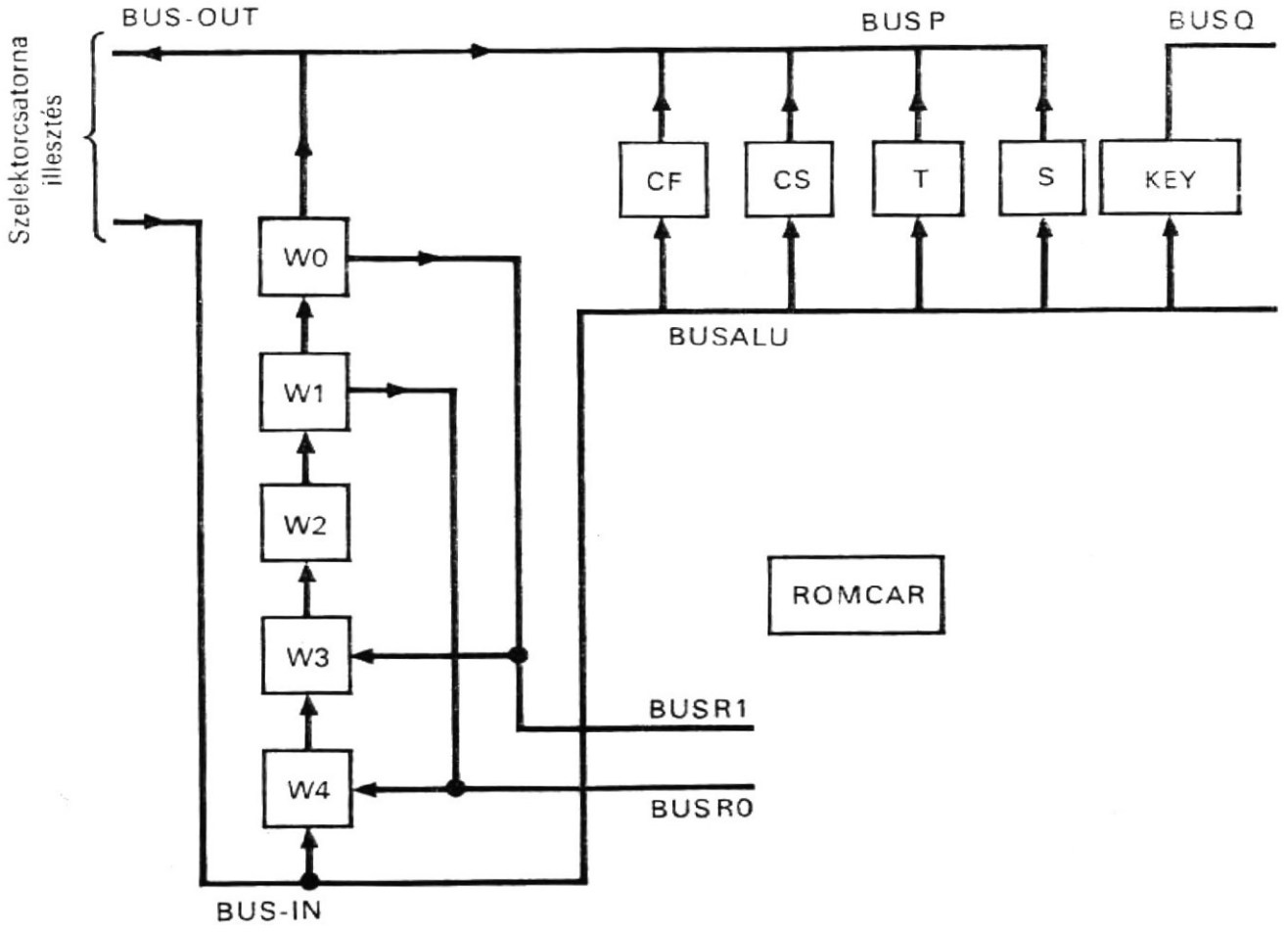
A megszakítási puffer formátuma a szelektorcsatornához

Bitek	Rendeltetés
0—3	Nincs felhasználva
4	Csatornavezérlés ellenőrzése „logout"-ra
5	Figyelmeztetés készülékvégre
6	Vég
7	PCI
8—18	Egységszám

tartalmát tárolja. Az EVSZ2-ben helyezkedik el a következő csatornaparancsszó, a CSPSZ címe, amely a csatornacímszóból (CAW—CCSZ, **C**hannel **A**ddress **W**ord: CAW) jövő pillanatnyi CPSZ cím, 8-cal megnövelve. Az EVSZ3 az íráskor az újratöltési címet tárolja az adatláncolás alatti felhasználásra. Az EVSZ4 a megszakítási puffert tárolja; ennek formátumát a 10.8. táblázatban láthatjuk. A puffer első byte-ja nullákat vagy jelzőbiteket tartalmaz; a második byte-ban annak a készüléknek az egységszáma van, amely pillanatnyilag a csatornán dolgozik. Az EVSZ5 az általános munkaterület.

10.4.2. Adatáram

A szelektorcsatorna külön regisztereket tartalmaz az adatok és a vezérlési információ tárolására, amint a 10.15. ábrán látható. Az S regiszter tárolja a CCW (CPSZ)-ből kapott főtáradatokat; ezáltal szükségtelenné válik az A regiszter betöltése, és ily módon megtakarítható az ehhez szükséges idő. Az S regiszternek két byte-ja van a három kiterjesztésbiten kívül. A T regiszter végzi az adatátviteli műveletek alatti byte-számlálást, és ezáltal gyorsan elvégezhető a szükséges byte-számlálás. A T regiszternek is két byte-ja van. Öt egy byte-os regiszterből áll a byte-puffer, amelynek feladata a főtár és az interface-vonalak közötti átmeneti tárolás. Ezek a regiszterek nyújtanak segítséget a helyi tár hozzáféréséhez az adatátvitel alatt. A CF csatornajelzőbit regiszter és a CS ellenőrzés- és állapotregiszter a csatorna vezérlésére szolgáló adatokat tárolja. Ezekkel gyors vezérlési műveletek végezhetők el, és ugyancsak



10.15. ábra
Szelektorcsatorna kialakítása az IBM/360 40-es modelljéhez

szükségtelenné teszik a helyi tárhozzáférést. A KEY (KULCS) regisztert, amely a csatorna tárvédelmi kulcsregisztere, a csatornacímszóban levő védelmi kulcs tárolására használják. A ROMCAR regiszter, amely a szelektorcsatorna csak olvasható tárának címregisztere, a csak olvasható tár gyors elérésére szolgál.

A D főtár tartalom-(puffer-) regiszternek viszont most ugyanazt a funkciót kell végrehajtania, mivel van elegendő idő a főtárműveletek alatt a D regiszter tartalmának mozgatására. A D regiszter tartalmát tehát az adatátvitelhez szükséges tárelérés alatt kell mozgatni.

A következő utasításokkal írhatjuk le a szelektorcsatorna előbb említett regisztereit és síneit:

Comment, a szelektorcsatorna felépítése (10.6)

```
Registers,      W0(P, 0—7),          $bytepuffer
                W1(P, 0—7),          $bytepuffer
                W2(P, 0—7),          $bytepuffer
                W3(P, 0—7),          $bytepuffer
                W4(P, 0—7),          $bytepuffer
Register,       T0(P, 0—7),
                T1(P, 0—7),
Casregister,    T=T0—T1,             $számlálóregiszter
Register,       SX(P, 5—7),          $3 címkiterjesztés-bit
                S0(P, 0—7),
                S1(P, 0—7),
Casregiszter,   S=SX—SO—S1,          $adat címe a főtárban
Register,       CF(0—7),             $csatornajelzőbit (flag) regiszter
                CF(CDA)=CF(0)        $adatláncolást jelző bit
                CF(CC)=CF(1)         $parancsláncolást jelző bit
                CF(SLI)=CF(2)        $hosszjelzéselnyomást jelző bit
                CF(SKIP)=CF(3)       $ugrás-skip-jelzőbit
                CF(YCH1)=CF(4)
                CF(YCH3)=CF(5)
                CF(RW)=CF(6)
                CF(RBD)=CF(7)
Register,       CS(0—7),             $csatorna-ellenőrzés és állapotregiszter
                CS(PCI)=CS(0)        $programvezérelt megszakításbit
                CS(WLR)=CS(1)        $rossz hosszúságú rekordbit
                CS(RGC)=CS(2)        $programellenőrzés-bit
                CS(PTC)=CS(3)        $védelemellenőrzés-bit
                CS(CDC)=CS(4)        $csatornaadat-ellenőrzés
                CS(CCC)=CS(5)        $csatornavezérlés-ellenőrzés
                CS(ICC)=CS(6)        $interface-vezérlés ellenőrzés
                CS(CC)=CS(7)         $láncolás ellenőrzés
Register,       KEY(P, 4—7),         $csatorna tárvédelem kulcsregiszter
Register,       ROMCAR(12—0),        $csak olvasható tár címregiszter a szelektor-
                                       csatornához
Bus,            BUSP(0—7, P),        $aritmetikai-logikai egység (ALU) P sín
                BUSQ(0—7, P),        $ALU Q sín
                BUSALU(0—7, P),      $ALU kimeneti sín
                BUSR0(0—7, P),       $az R sín első byte-ja
                BUSR1(0—7, P),       $az R sín második byte-ja
```

A byte-puffer a következőképpen látja el a bejövő és kimenő byte-ok átmeneti tárolását. Olvasás művelet esetén a be-, ill. kimeneti készüléktől kapott byte-ot a

W4 puffer tárolja és ezután a W0 puffernek adja át, majd fogadja a következő bejövő byte-ot és továbbadja a W1 puffernek. Ez a két byte azután átvivődik a W1 és W0 pufferből a D regiszterbe, ahonnan a főtárba kerülnek, az S regiszter által meghatározott helyre. Írás művelet esetén két byte-ot olvasnak ki a főtárból, és a W3 és W4 pufferekbe írják át őket; ezután a W0 és W1 pufferekbe kerülnek. Ezt a két byte-ot küldik aztán tovább byte-onként a vezérlőegységhez.

A csatornacímszó (CAW—CCSZ) által szolgáltatott tárvédelmi (TV) kulcsot a KEY jelű csatornakulcs-regiszter tárolja. Valahányszor a szelektorcsatorna működése közben adatbyte tárolása miatt hozzáférés történik a főtárhoz, a tárvédelem helyi tárolójából (TVHT) beolvassák a TV kulcsot a TVHT adatregiszterbe. A két regiszter tartalmát összehasonlítják; ha a kettő nem egyezik meg, hibajelzés jön létre.

10.4.3. Adatátvitel (szolgálatkérés)

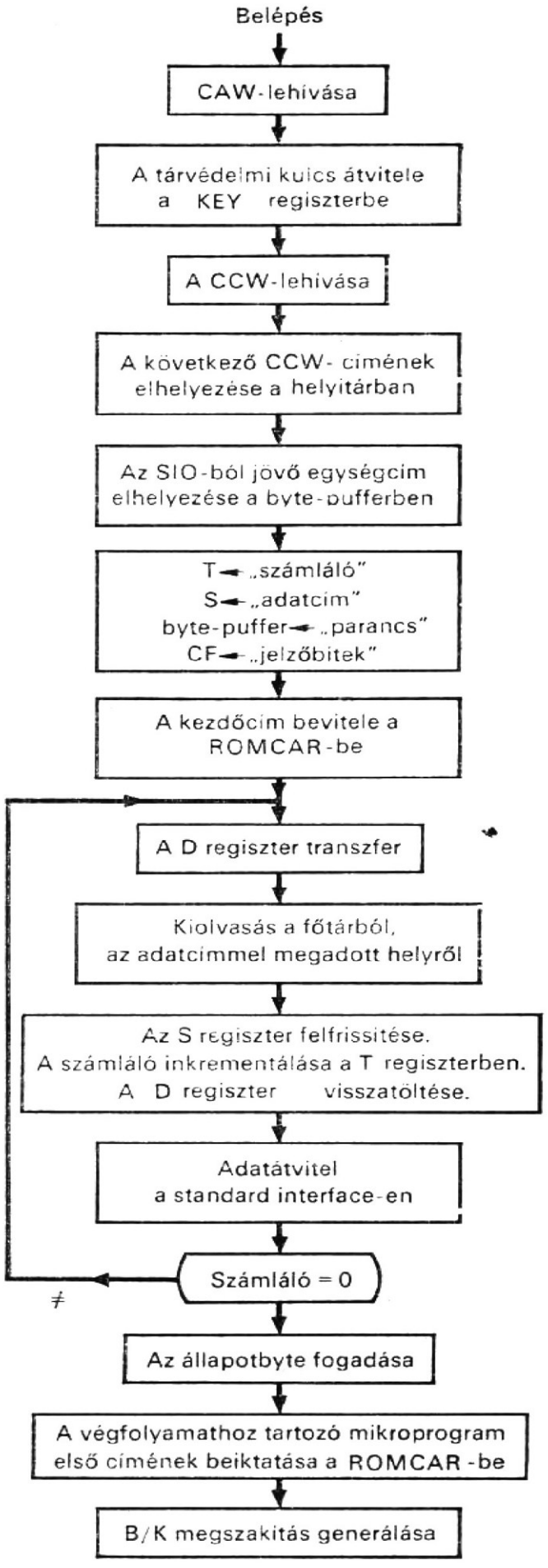
Az adatátvitel, más elnevezéssel szolgálatkérés bemutatására illusztrációképpen leírjuk a szelektorcsatorna által végzett írási műveletet. Az írási művelet folyamatábráját a 10.16. ábra mutatja. A CAW (CCSZ) csatornacímszót lehívják, a TV tárvédelmi kulcsot eltávolítják és beillesztik a csatornakulcs-regiszterbe. Ezután lehívják a CCW (CPSZ) csatornaparancsszót, és elhelyezik a következő CCW (CPSZ) címét a helyi tárolóban, az UCW (EVSZ) egységvezérlő szóban. A START I/O (B/K start) utasításból meghatározott egységcímet a byte-pufferben helyezik el a kezdeti kiválasztás alatt, a be-, ill. kimeneti vezérlőegységhez való továbbítás céljából.

A CCW (CPSZ) tartalmát most szétosztják a regiszterek között, az átviendő byteszámot a T regiszterbe, az adatcímet az S regiszterbe, a parancsot a W byte-pufferbe, továbbá a jelzőbiteket a CF csatornajelzőbitek regiszterbe viszik a soron következő átviendő adat átviteli műveletének vezérlése céljából.

Ez az írási művelet a ROMCAR csatornacím-regiszterben levő megfelelő kezdőcímmel indul, amely azután átveszi a vezérlést a csak olvasható tár ROMADDR címregiszterétől és elágazik a szelektorcsatorna szolgálatkérési (adatátviteli) mikroprogramjához. Ez a mikroprogram kiolvassa a D regisztert, kiolvassa az S regiszter által megcímzett adatot, felfrissíti az S regisztert és a T regiszterben levő számlálót, és visszatölti a D regisztert, majd ezután a B/K jelvezetékekre történő adatátvitel következik. A csatorna működése a további szolgálatkérési ciklusokkal folytatódik, és a csatorna ezekre mindaddig válaszol, amíg a byte-számláló 0 nem lesz, ezután a be-, ill. kimeneti készülék azt az utasítást kapja, hogy álljon le. Mindezek a műveletek standard interface megismert jelsorozatát használják. Az adatátvitelek vezérlését általában inkább sorrendi logikával végzik, mint mikroprogramozott vezérléssel.

Amikor befejezési állapotjel érkezik, fogadni kell az állapotbyte-ot és el kell helyezni a megfelelő címet a csak olvasható tár csatornacím-regiszterében (ROMCAR), a végfolyamathoz tartozó mikroprogramba való belépés céljából. Ezután normál be-, ill. kimeneti megszakítás generálódik.

Minthogy az előbbiekben leírt műveletek többségét inkább sorrendi logikával vezérlik, mint mikroprogramokkal, messzemenően kevesebbszer avatkoznak be a CPU programba, tehát a be-, ill. kimeneti készülékek adatkiszolgálása is gyorsabban megy végbe.



10.16. ábra
A szelektorcsatornával végzett írási műveletet bemutató folyamatábra

10.4.4.
Be-, ill. kimeneti (I/O) start

A be-, ill. kimeneti start (start I/O=SIO) utasítás a csatornával olvasás, írás, visszafelé olvasás, vezérlés vagy érzékelés műveletet hajtat végre a SIO utasításban specifikált készülékkel. Ez a következő funkciókat végzi:

— kiválasztja a csatornát, a vezérlőegységet és a be-, ill. kimeneti készüléket;
— kiadja a parancsot;
— betölti a csatornaregiszterekbe a CCW csatornaparancsszóból kapott csatornavezérlési információt, és ezzel képessé teszi a csatornát az adott műveletnek a specifikált készüléken való végrehajtására;
— beállítja a feltételkódot az Y2 és Y3 statokban.

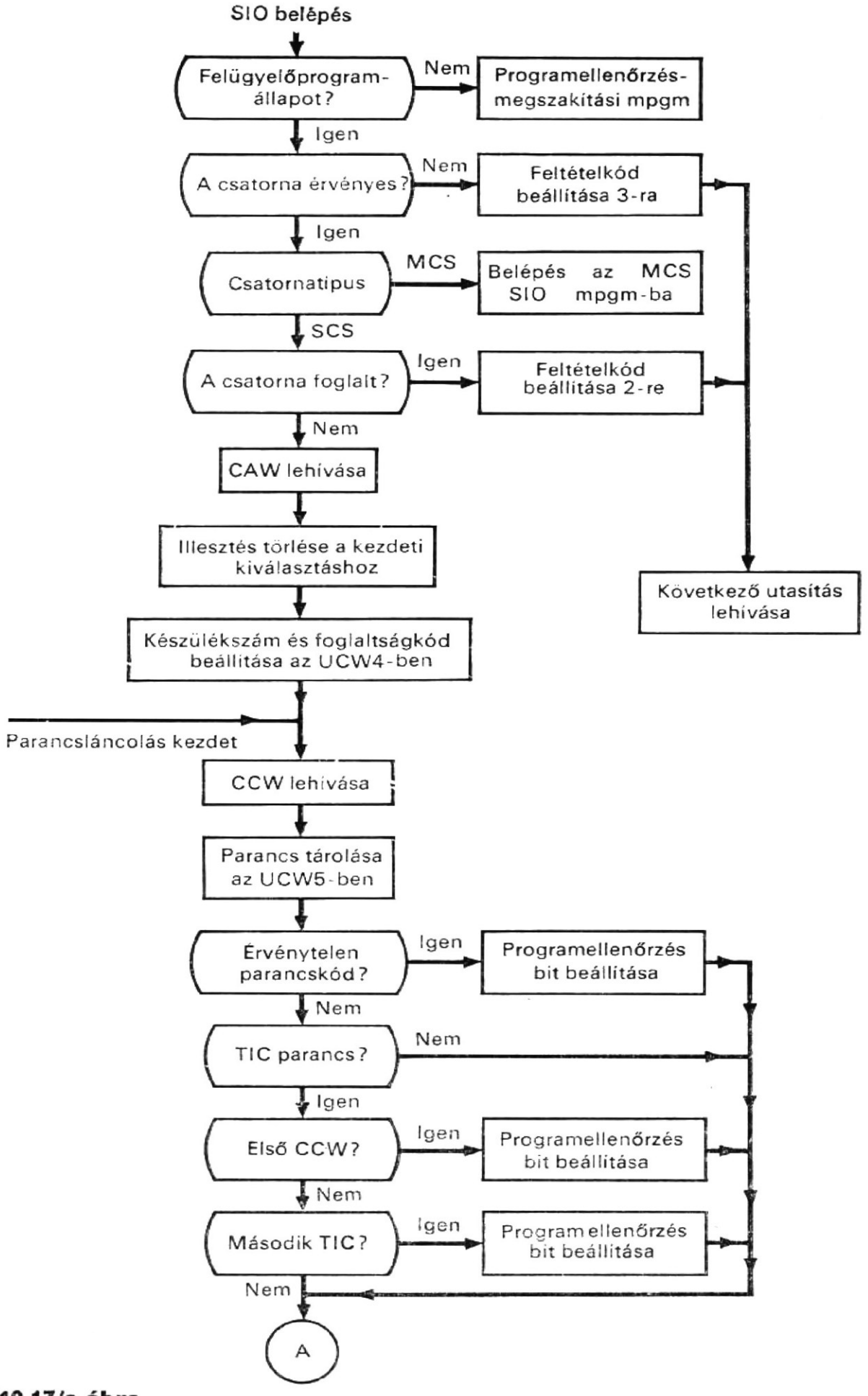
Ha a SIO sikeresen indult, a feltételkód 0 értéket kap; a CPU lehívja a következő utasítást. A csatorna ekkor a CPU programmal párhuzamosan működik, csak abban az esetben szakítva meg a CPU programot, ha a főtárhoz kell hozzáférés, vagy ha adatbyte-okat kell eltávolítani a csatornapufferből.

A SIO művelet folyamatábráját a 10.17. ábra mutatja. Amint itt látható, ha a CPU nincs felügyelő (supervisor) állapotban, a programellenőrzéshez tartozó megszakítási mikroprogram lép be, és a SIO végrehajtását felfüggeszti. Ha a CPU felügyelő állapotban van, de a csatorna nem érvényes szelektorcsatorna, a feltételkód a 3 értéket kapja, és a következő utasítást hívják le. Ha a csatorna érvényes szelektorcsatorna, de egy korábban indított művelet vagy folyamatban levő megszakítás foglalja el, a feltételkód 2 értéket kap, és a következő utasítást hívják be. Ha az érvényes szelektorcsatorna szabad, a csatornacímszót (CAW) hívják le, és ennek alapján meghatározzák a csatornaparancsszót (CCW), amely a vezérlési információnak (parancskód, főtárbeli adatcím, byte-szám, jelzőbitek) a csatornaregiszterekbe való betöltésére szolgál. Az interface-t törlik a kezdeti kiválasztás céljából. Amikor ez megtörtént, a helyi tár megszakítási pufferjének UCW4-be, ill. EVSZ4-jében elhelyezik a készülékszámot és a foglaltság-kódot annak jelzésére, hogy a csatorna ettől kezdve foglalt (műveletet végez).

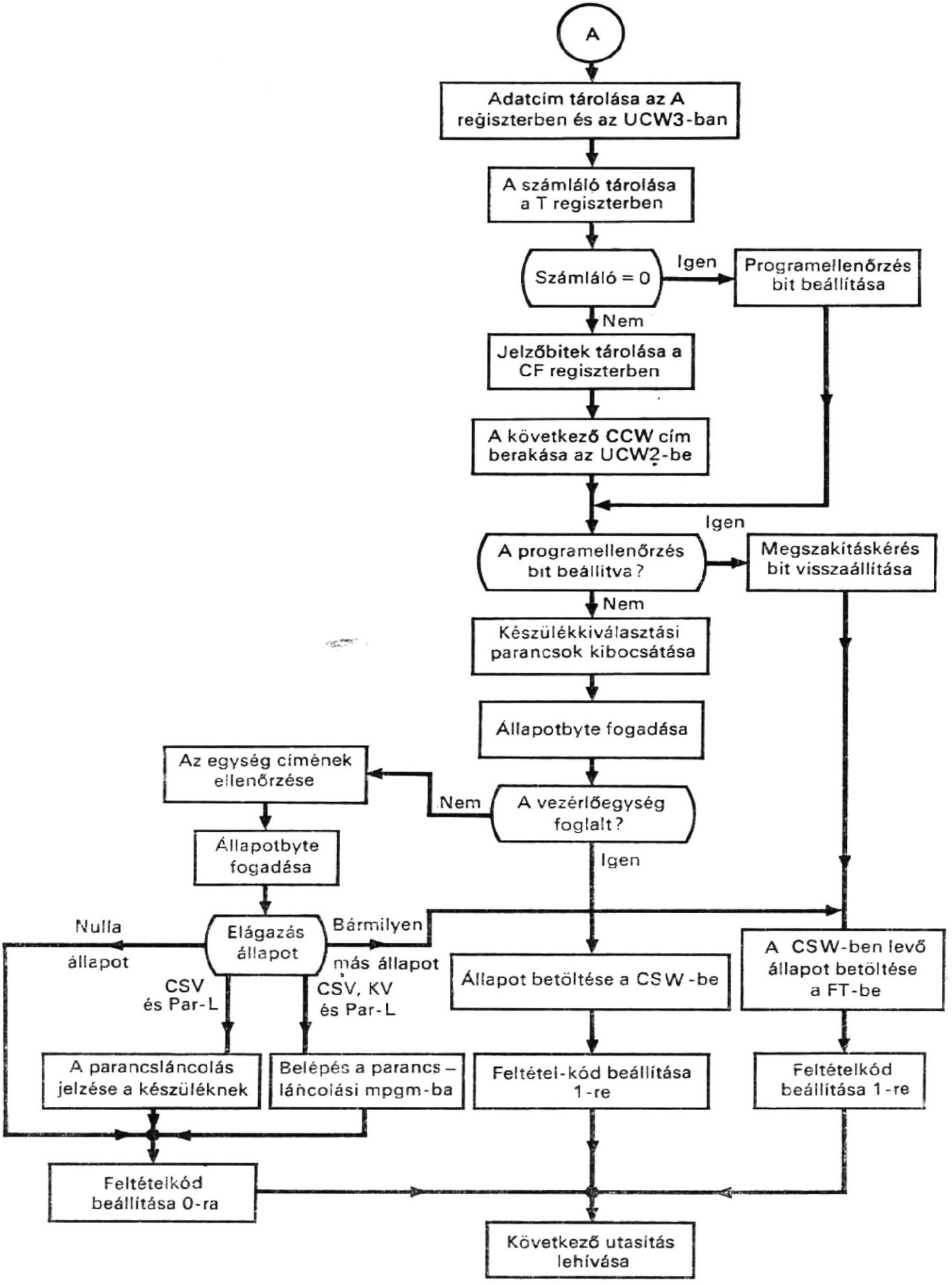
A CCW-t a főtárból olvassák ki. A parancskódot a helyi tárban az UCW5-ben tárolják, és érvényességét ellenőrzik. Ha érvénytelen, ezt a programellenőrzési bit beállításával jelzik. Ha érvényes, megvizsgálják, vajon TIC parancs-e. Ha TIC parancs és az első CCW-től jön, a programellenőrzési bitet vezérlik, mivel TIC parancs első csatornaparancsszóként nem specifikálható SIO utasításban. A TIC parancs csak akkor lehet érvényes, ha parancsláncolásból ered (vagyis nem az első CCW-ből), de akkor nem szabad másik TIC parancsot specifikálnia. Ha érvényes parancsról vagy pedig egy helyes TIC parancsról van szó, az adatcímet az A regiszterben és az UCW3-ban tárolják a helyi tárban; a számlálás eredményét a T regiszter tárolja. Ha a számlálás értéke 0, a programellenőrzés bit 1 értéket kap. A jelzőbiteket a CP jelzőbitregiszter tárolja.

Ezek után vizsgáljuk meg a programellenőrzési bitet. Ha 1 értéket kapott, akkor a megszakításkérési bit visszaállítását idézi elő, a csatornaállapot-szó (CSSZ) állapot részének értékét betöltik a főtárba, és a feltételkód 1 értéket kap. Azzal, hogy a feltételkódnak 1 értéket adnak, arról értesítik a központi egységet, hogy a SIO utasítást nem sikerült végrehajtani, a sikertelenség okáról a CSSZ állapot felülvizsgálatával lehet meggyőződni.

Ha nincs programellenőrzés, a csatorna parancsokat és címet bocsát ki a készülék kiválasztására, és egy állapotbyte-ot kap. Ha a csatornaegység foglalt, az állapotinformációt betöltik a CSSZ-be, a feltételkód 1 értéket kap, és lehívják a következő utasítást. Ha a csatornaegység nem foglalt, ellenőrzik az egység címét, és belép egy állapotbyte. Ha az állapotbyte nulla, a feltételkód 0 értéket kap, és lehívják a következő


SIO belépés
Felügyelőprogram-állapot?
Nem
Programellenőrzés-megszakítási mpgm
Igen
A csatorna érvényes?
Nem
Feltételkód beállítása 3-ra
Igen
Csatornatipus
MCS
Belépés az MCS SIO mpgm-ba
SCS
A csatorna foglalt?
Igen
Feltételkód beállítása 2-re
Nem
CAW lehívása
Illesztés törlése a kezdeti kiválasztáshoz
Következő utasítás lehívása
Készülékszám és foglaltságkód beállítása az UCW4-ben
Parancsláncolás kezdet
CCW lehívása
Parancs tárolása az UCW5-ben
Érvénytelen parancskód?
Igen
Programellenőrzés bit beállítása
Nem
TIC parancs?
Nem
Igen
Első CCW?
Igen
Programellenőrzés bit beállítása
Nem
Második TIC?
Igen
Programellenőrzés bit beállítása
Nem
A

10.17/a ábra



Rövidítések:
CSV: Csatornavég
Par-L: Parancslánc
KV: Készülékvég

10.17. ábra
A SIO utasítás végrehajtását bemutató folyamatábra

utasítást. Ha az állapot csatornavég jelet tartalmaz, specifikált parancsláncolással, a csatorna azt jelzi a készüléknek, hogy parancsláncolás következik akkor, amikor a csatorna megkapja a készülékvég jelet. A feltételkódot 0-ra állítják, és lehívják a következő utasítást. Ha az állapot mind csatornavég, mind készülékvég jelet tartalmaz, specifikált parancsláncolással, a parancsláncolási mikroprogram lép be; a feltételkód 0 értéket kap, és lehívják a következő utasítást. Ha az állapot bármilyen más állapot, a CSSZ-ben levő állapotot betöltik a főtárba; a feltételkód 1 értéket kap, és lehívják a következő utasítást.

10.4.5.
Egyéb be-, ill. kimeneti utasítások

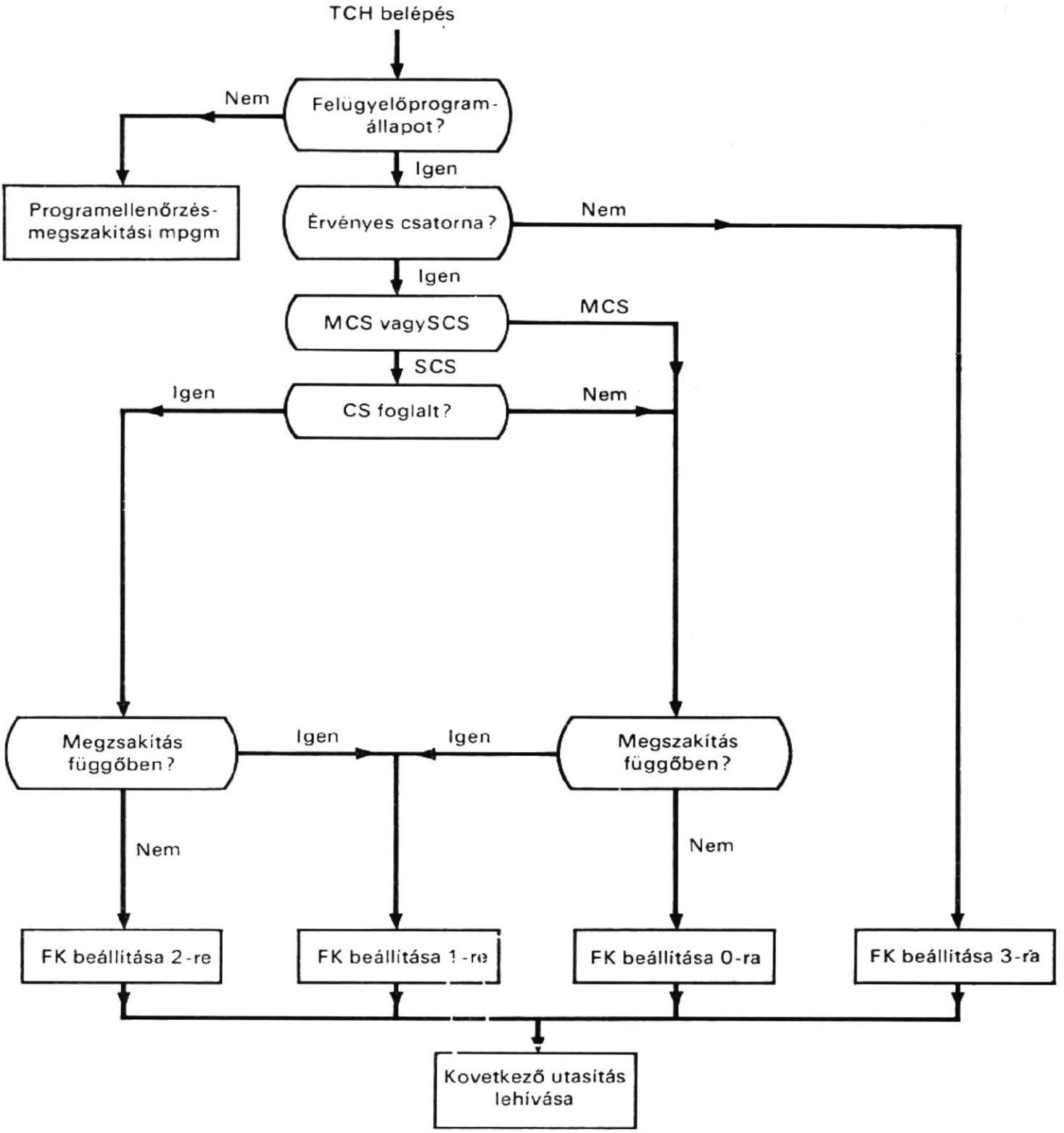
A TCH csatornatesztelés (**T**est **CH**annel: TCH) utasítás egy megcímzett csatorna állapotát határozza meg, és beállítja a feltételkódot a PSSZ(PSW) programállapot-szóban, amint az a 10.3. táblázatban látható. A TCH műveletet a 10.18. ábrán levő folyamatábrán láthatjuk. A művelet során először megvizsgálják a felügyelőprogram (supervisor) állapotban kibocsátott utasítás feltételét; ez a PSW (PSSZ) 15. bitjének tesztelésével történik. Ha a PSW (PSSZ) 15. bitje 1, ez feladatállapotot jelez, és az ellenőrzést végző megszakítási mikroprogram veszi át a vezérlést. Ha a PSSZ 15. bitje 0, (felügyelőprogram állapot) megvizsgálják a csatorna számának érvényességét. Ha a csatorna száma nem érvényes, a feltételkód 3-as értéket kap, ami azt jelzi, hogy nincs elérhető csatorna.

Ha a csatornát érvényesnek találják, a végrehajtás elágazik aszerint, hogy a csatorna szelektor- vagy multiplexcsatorna-e. Ha az szelektorcsatorna (SCS), megvizsgálják a helyi tárban levő egységvezérlő szót (EVSZ) annak megállapítására, hogy foglalt-e a csatorna. Ha a csatorna nem foglalt, megvizsgálják az SCS megszakításkérési regiszterét. Ha ez a regiszter 1, a feltételkód 1-es értéket kap annak jelzésére, hogy egy megszakítás függőben van. Egyébként a megszakításkód 0 értéket kap, jelezve, hogy a csatorna és az egység rendelkezésre áll. Abban az esetben, ha a csatorna egy benne tárolt megszakítás miatt foglalt (azaz egy megszakítás függőben van), a megszakításkód 1-es értéket kap. Ha a csatorna azért foglalt, mert dolgozik (nincs függőben levő megszakítás), a feltételkód 2-es értéket kap. Mindegyik esetben lehívják a következő utasítást. A csatornatesztelés utasítást rendszerint egy feltételes utasítás követi, amely — a feltételkódtól függő — elágazást idéz elő a programban.

A szelektorcsatorna HIO (**H**AIT **IO**) és TIO (**T**EST **IO**) utasításai hasonlóak a multiplexcsatorna ezen utasításaihoz; ezeket a következő fejezetben fogjuk megtárgyalni.

10.5.
A multiplexcsatorna

Az IBM System/360 40-es modellje közepes kategóriájú digitális számítógép. Abból az elgondolásból, hogy a rendszer kevésbé legyen költséges, a multiplexcsatorna (MCS) osztozik valamennyi CPU regiszteren és sínen, kivéve néhány járulékos regisztert, amelyek az átmeneti tárolást és az illesztést kezelik. Az MCS-t működtető és vezérlő mikroprogram megkülönböztetett része a multiplexcsatornának. Az MCS két üzemmódban működik:



10.18. ábra
A csatornateszt TCH műveletet bemutató folyamatábra

— burst- (lökésszerű) üzemmódban, ekkor egyetlen nagy sebességű készüléket vezérel, és
— byte-üzemmódban (vagy a multiplex üzemmódban), ekkor több kis sebességű készülékhez tartozik.

Ebben az üzemmódban, ha az MC átvette a vezérlést, a CPU leválik a csatornáról. Ekkor a CPU és az MCS szimultán működik, kivéve azt az időt, amikor az MCS főtár hozzáférést végez. Improduktív (overhead) idő még a CPU és az MCS átkapcsolása közötti idő is. A burst- üzemmódban az MCS összekapcsolva marad a központi egységgel a teljes be-, ill. kimeneti művelet idejére; ekkor nincs egyidejű működés közöttük.

Négy be-, ill. kimeneti utasítás van:

— SIO (Start I/O: B/K start),
— TIO (Test I/O: B/K teszt),
— HIO (Halt I/O: B/K állj),
— TCH (Test Channel: csatornateszt).

Ezen utasítások végrehajtását részben a CPU mikroprogramja, részben az MCS mikroprogramja végzi. A továbbiakban a B/K utasításokat végrehajtó csatornaműveleteket írjuk le, miután bemutattuk a csatornavezérlő szót.

10.5.1.
A csatornavezérlő szó

Az MCS mindegyik alcsatornája egy be-, ill. kimeneti készüléket vezérel. Egy be-, ill. kimeneti készülék működésének fenntartásához szükséges összes információt az UCW, az egységvezérlő szó (EVSZ) tartalmazza. Az alcsatornákra vonatkozó UCW-ket a multiplex (mpx) tárolja. Az UCW formátuma a 10.9. táblázatban található. Mindegyik UCW 16 byte-ot foglal el, amelyekből 6-ot nem használnak föl. Mivel maximálisan 128 alcsatorna lehet, az mpx tárnak 1024 félszó a kapacitása. Az UCW gyűjti össze a készülék címét, a csatornaparancsszó címét, a tárvédelmi kulcsot, a parancskódot, a jelzőbiteket, az adatcímet, a byte-számot és az állapotot, ugyanazokból a forrásokból, mint a szelektorcsatorna.

10.9. táblázat

Az egységvezérlőszó (EVSZ) formátuma a multiplexcsatornánál

Bitek	Rendeltetés
EVSZ0(0—15)	Számláló
EVSZ2(0—15)	Adatcím, kivéve az átvitel végét
EVSZ2(0—7)	Az egység száma az átvitel végén
EVSZ2(8—15)	Az egységállapot az átvitel végén
EVSZ4(0—4)	Jelzőbitek
(0)	Láncolt adatok címe (CDA)
(1)	Parancslánc (CC)
(2)	Hosszlenyomás jelzése (SLI)
(3)	Ugrás (skip)
(4)	Programvezérelt megszakítás (PCI)
EVSZ4(5—7)	Műveleti- (op-) kód
EVSZ4(8)	Nulla számlálóindikátor
(9)	Vég állapot elérve indikátor
(10—15)	Az adatcím kiterjesztése
EVSZ6(0—15)	Új csatornaparancsszó CCW címe
EVSZ8(0, 4, 5, 7)	Nincs felhasználva
(1)	Rossz hosszúságú rekord (WLR)
(2)	Programellenőrzés
(3)	Védelemellenőrzés
(6)	Illesztésvezérlés-ellenőrzés (ICC)
(8—15)	Új CCW cím kiterjesztés

10.5.2. Csatornatesztelés

A multiplexcsatorna esetén a csatornateszt, a TCH utasítás működése hasonló a szelektorcsatornáéhoz, amint azt a 10.18. folyamatábrán bemutattuk. Röviden, ha a CPU felügyelőprogram állapotban van, a csatorna érvényes és a kívánt csatorna MCS, akkor tesztelik a multiplexcsatorna megszakításkérési regiszterét. Ha ez a regiszter 1, a feltételkód (l. 10.3. táblázat) 1 értéket kap, annak jelzésére, hogy van egy függőben levő megszakítás; egyébként pedig a feltételkód 0 értéket kap jelezve, hogy a csatorna és a készülék rendelkezésre áll.

10.5.3. Be-, ill. kimenet állj

A HIO (B/K állj) utasítás közvetlenül leállítja a készüléket, vagy akkor állítja le, amikor legközelebb szolgálatkéréssel jelentkezik. Ez privilegizált művelet. A HIO utasítás a multiplexcsatornában csak abban az esetben működik, ha a csatorna byte-üzemmódban dolgozik. A HIO működését a 10.19. folyamatábrán láthatjuk. Először megvizsgálja, hogy az állapot felügyelőprogram állapot-e, a csatorna érvényes-e, és hogy az alcsatorna foglalt-e. Ha a csatorna nem érvényes, a feltételkód (l. 10.3. táblázat) 3 értéket kap, és lehívják a következő utasítást.

Ha az alcsatorna foglalt vagy megszakítást tartalmaz (vége állapotot ért el), a feltételkód 0 értéket kap, és lehívják a következő utasítást.

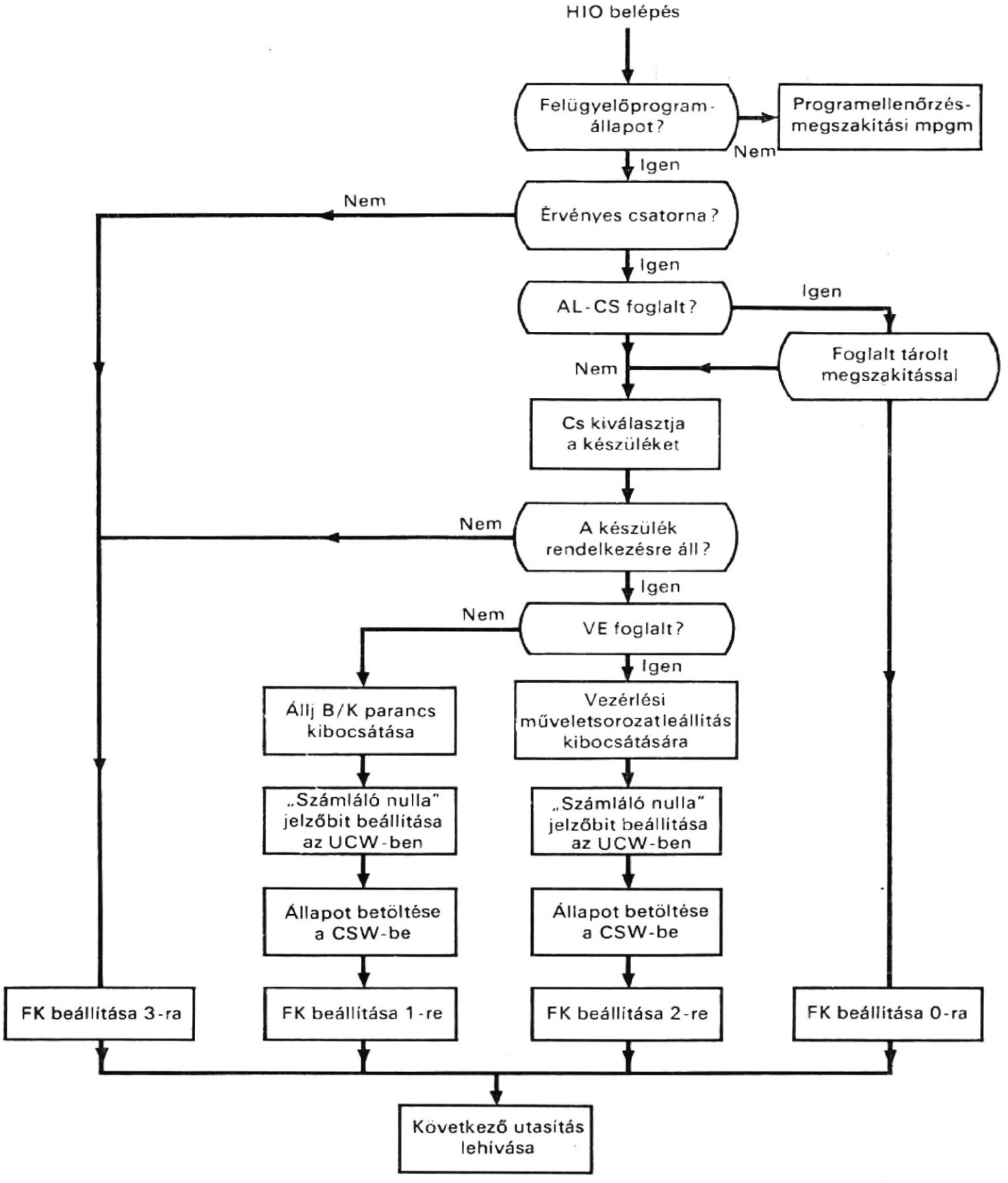
Ha az alcsatorna nem foglalt és nem tartalmaz megszakítást, művelet van folyamatban a kiválasztott alcsatornán. A készüléket hozzákapcsolják a csatornához, és működését leállítják. Ha az alcsatorna nem foglalt, a készüléket akkor is hozzákapcsolják a csatornához, valamilyen lehetséges tárolt készülékvég megszakítás okának megvizsgálása végett.

Ebben az esetben, ha a készülék kiválasztás azt jelzi, hogy a készülék nem áll rendelkezésre, a feltételkód 3-as értéket kap, és lehívják a következő utasítást.

Ha a készülék rendelkezésre áll, és ha a vezérlőegység nem foglalt, a csatorna bocsát ki egy HIO parancsot a készülék működésének leállítására. Beállítják az UCW-ben a számlálásnulla jelzőbitet; az állapotot a CSSZ csatornaállapot-szó tárolja, és a feltételkód 1-es értéket kap. Ha a készülék rendelkezésre áll, és ha a vezérlőegység foglalt, a csatorna egy stop vezérlősorozatot bocsát ki a készülék leállítására akkor, amikor legközelebb szolgálatkéréssel jelentkezik. Ismét beállítják a számlálásnulla jelzőbitet; az állapotot a CSSZ-ben tárolják, és a feltételkód 2-es értéket kap. Mindegyik esetben lehívják a következő utasítást.

10.5.4. A be-, ill. kimenet tesztelése

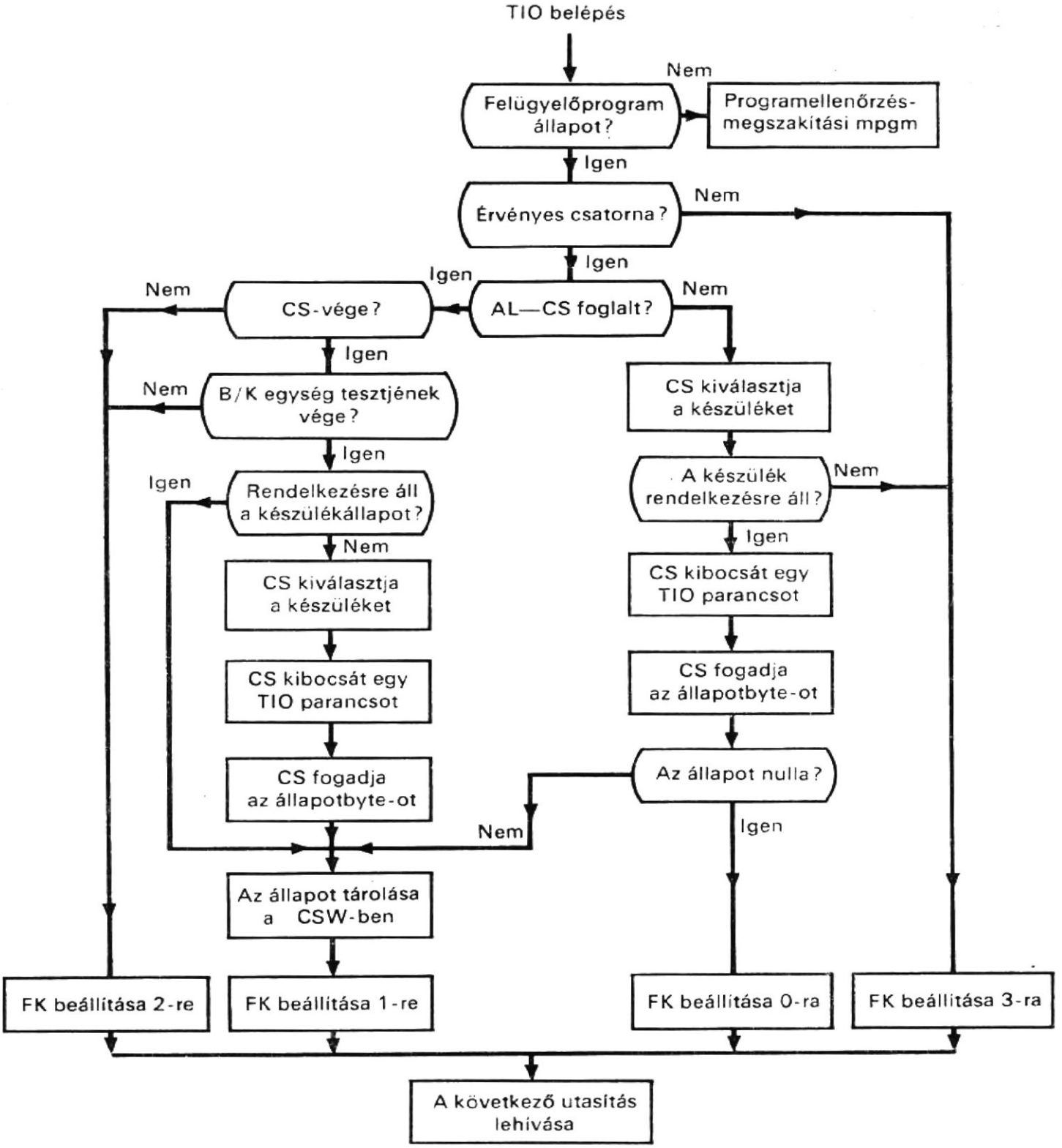
A TIO (B/K teszt) utasítás megvizsgálja egy be-, ill. kimeneti készülék állapotát, és ennek megfelelően beállítja a feltételkódot. A TIO működését a 10.20. folyamatábrán mutatjuk be. A működés annak vizsgálatával kezdődik, hogy a CPU felügyelőprogram állapotban van-e. Ha nincs felügyelőprogram állapotban, a CPU behozza a programellenőrzés megszakítási mikroprogramot; az utóbbi a privilegizált működésbitnek 1



10.19. ábra
A HIO (állj B/K) műveletet bemutató folyamatábra

értéket ad a pillanatnyi programállapotszó (PSW) megszakításkódjában, elhelyezi ezt a PSW-t a főtárban, és lehívja az új PSW-t a főtárból.

Ha a CPU felügyelőprogram állapotban van, legelőször megvizsgálják a csatorna érvényességét. Ha a csatorna nem érvényes vagy nincs működő állapotban, a feltételkód (l. 10.3. táblázat) 3-as értéket kap, és lehívják a következő utasítást.



10.20. ábra
A TIO (B/K teszt) műveletet bemutató folyamatábra

Ha a CPU felügyelőprogram állapotban van, a csatorna érvényes, és az alcsatorna nem foglalt, akkor a csatorna kiválasztja a be-, ill. kimeneti készüléket, kibocsát egy B/K tesztparancsot (nem a TIO parancsot), és fogadja az állapotbyte-ot a készüléktől. Ha az állapotbyte azt jelzi, hogy a készülék nem áll rendelkezésre, a csatornaállapot-szót tárolják, a feltételkód 1-es értéket kap, és lehívják a következő utasítást. Ha a készülék rendelkezésre áll, a feltételkód 0 értéket kap, és lehívják a következő utasítást.

Ha a csatorna érvényes, de az alcsatorna foglalt, megvizsgálják a vezérlőegységtől jövő csatorna vége jelet. Ha nincs csatornavég jel, ez azt jelzi, hogy a készülék éppen dolgozik, és a feltételkód 2-es értéket kap. Ha a B/K tesztparancs elérte a véget, és ha a

készülékállapot az átvitel végén rendelkezésre áll az alcsatornában, az állapotot tárolják a CSSZ csatornaállapot-szóban, és a feltételkód 1-es értéket kap. Ha a készülékállapot nem áll rendelkezésre az alcsatornában, a csatorna kiválasztja a készüléket, kibocsát egy B/K tesztparancsot, és fogadja az állapotbyte-ot a készüléktől. Az állapotot a CSSZ-ben tárolják és a feltételkód 1-es értéket kap. Mindegyik esetben lehívják a következő utasítást.

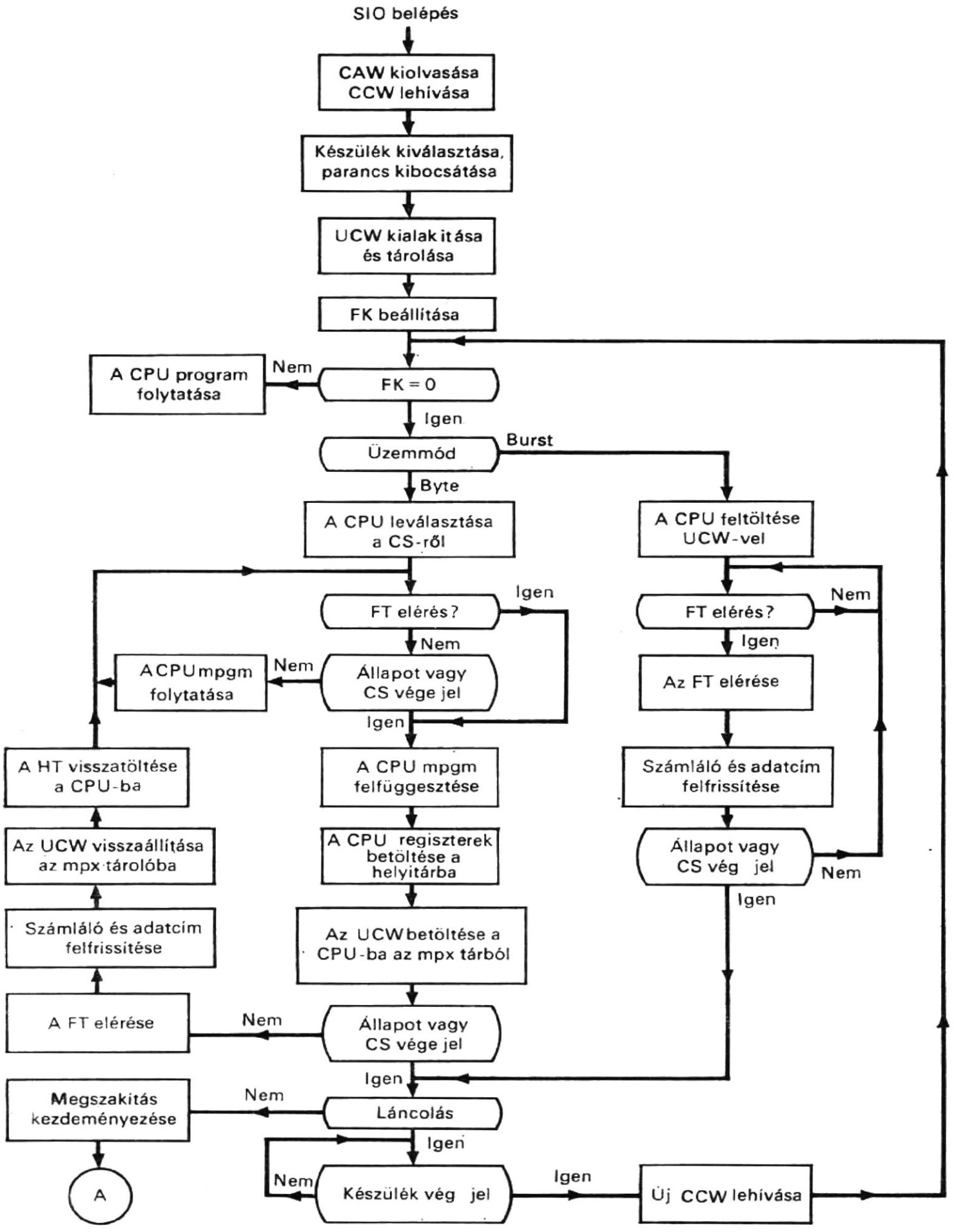
10.5.5. Be-, ill. kimeneti start

A SIO (B/K start) utasítás olvasás, írás, olvasás hátra, vezérlés vagy érzékelés műveletet kezdeményez a megcímzett be-, ill. kimeneti készüléknél, a CPSZ csatornaparancsszó 0...7 bitjében levő parancskód szerint. A szelektorcsatornában végrehajtott SIO-hoz hasonlóan, a multiplexcsatornában is a SIO választja ki a csatornát, a vezérlőegységet és be-, ill. kimeneti készüléket. Ez bocsátja ki a parancsot. Ez alakítja ki az egységvezérlő szót, és helyezi azt el az mpx tárban. Ez állítja be a megfelelő feltételkódot az Y2 és Y3 statokban. Ha az MCS byte-üzemmódban van, a csatorna leválik a CPU-ról és a központi egység lehívja a legközelebbi utasítást a sorozatban.

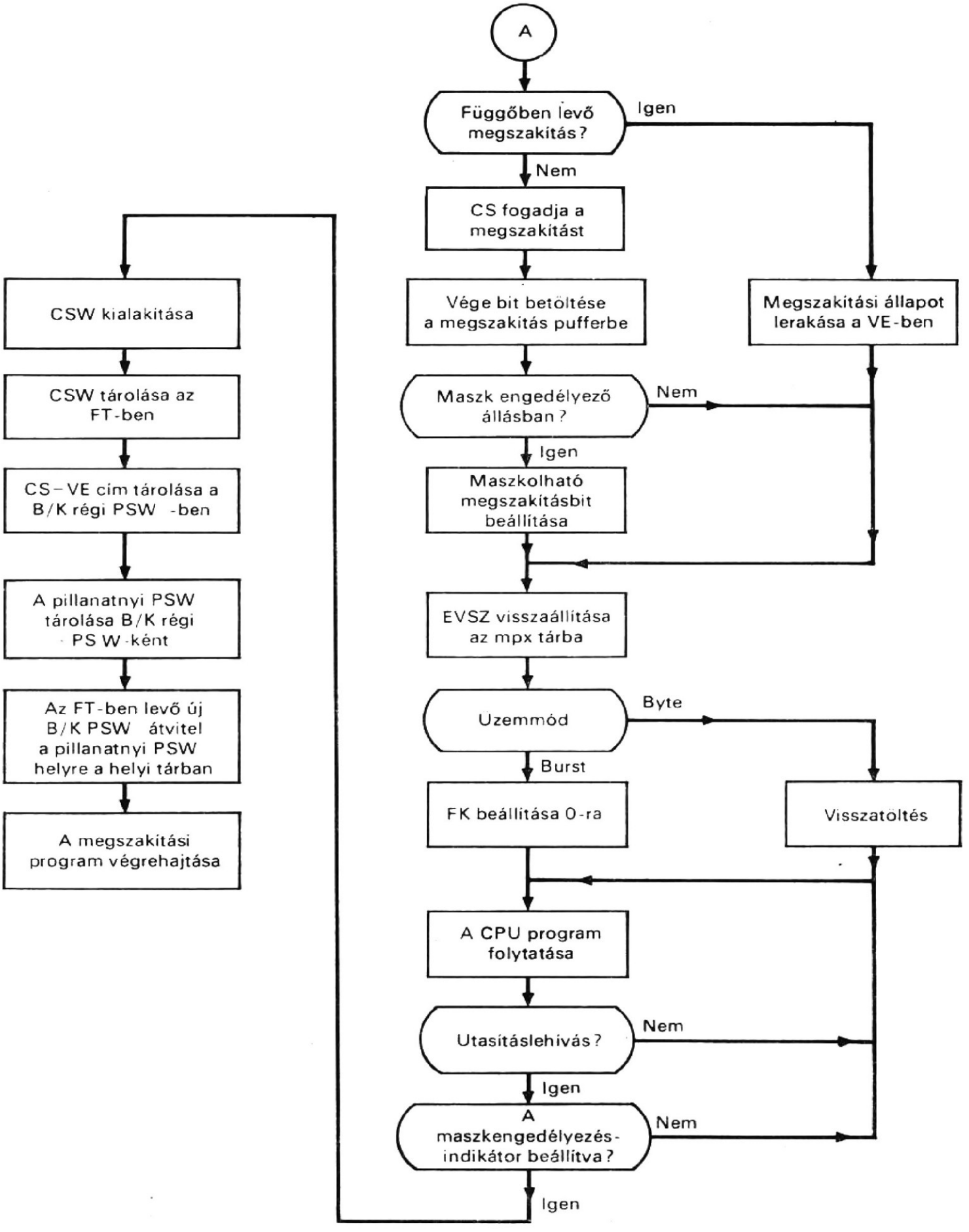
A SIO műveletet a 10.21. folyamatábrán láthatjuk. A CPU kiolvassa a főtárból a CCSZ csatornacímszót. A CCSZ adja meg azt a címet, amelyről a főtárból le lehet hívni az első CPSZ csatornaparancsszót. A CPSZ specifikálja a végrehajtandó műveletet, a felhasználandó főtárterületet, a művelet végrehajtásakor elvégzendő tevékenységet, és az átviendő byte-ok számát. Ezután kiválasztják a készüléket, és a CPSZ-ból átviszik a parancsot a készülékhez. Egy EVSZ egységvezérlő szót alakítanak ki, és elhelyezik az mpx tárban. Ha a kiválasztott készülék byte-üzemmódban dolgozik, a feltételkód 0 értéket kap annak jelzésére, hogy a csatorna rendelkezésre áll és nem foglalt. A CPU leválik a csatornáról, és a csatornával párhuzamosan dolgozik. A csatorna most csak akkor szakítja meg a központi egység programját, ha adatot akar tárolni, vagy lehívni a főtárba(ból).

A byte-üzemmód esetén, amikor a vezérlőegységnek adatátvitelre van szüksége (szolgálatkérés), pl. hozzáférés a főtárhoz, mindig mikroprogram-megszakítás jön létre. Ez a megszakítás felfüggeszti a CPU mikroprogramját, és egy mozgatást (dump) kezdeményez, amely az összes CPU regiszter tartalmát elhelyezi a helyi tárolóban. A mozgatás lehetővé teszi a CPU regiszterek számára, hogy csatornaként működjenek a vezérlőegységbe vagy onnan jövő adatszolgáltatásban. Most az alcsatorna adat és vezérlési információját az mpx tárban levő egységvezérlő szóból betöltik a CPU regiszterekbe. Az adatszolgáltatást a főtárhoz való hozzáféréssel hajtják végre, felhasználva az egységvezérlő szóból kapott adatcímet. Az adatokat a B/K vezérlőegységbe (vagy egységből) viszik át, és felújítják az EVSZ számlálóját és címmezőit. A felújított EVSZ-t az mpx tárban helyezik el. Ezután egy visszamozgatás (undump) megy végbe, amelynek során visszaállítják a CPU regisztereket a mikroprogram-megszakítást megelőző állapotukba. Befejezik egy adatszegmens átvitelét (szolgálatkérés befejeződik), és folytatódik a felfüggesztett CPU utasítás végrehajtása. A további adatszegmenseket a be-, ill. kimeneti készülék és a főtár közötti átvitel esetén hasonló módon kezelik, minden egyes átvitelt egy-egy B/K megszakítás kezdeményez, egészen addig, amíg csatornavég jelet nem érzékel a csatorna.

A burst-üzemmód esetén az alcsatorna adat és vezérlési információját ugyancsak az mpx tárban levő egységvezérlő szóból töltik be a CPU regiszterekbe. A központi egység nem válik le a csatornáról, hanem folytatja működését, úgy, mint a csatorna. Valahányszor készen állnak adatok a készülékből való átvitelre, a CPU hajtja végre az adatszolgáltatást, frissíti föl a számlálót és az adatcímmezőket az egységvezérlő szóban.



10.21. ábra
A SIO műveletet bemutató folyamatábra


A
Függőben levő megszakítás?
Igen
Nem
CS fogadja a megszakítást
Vége bit betöltése a megszakítás pufferbe
Megszakítási állapot lerakása a VE-ben
Maszk engedélyező állásban?
Nem
Igen
Maszkolható megszakításbit beállítása
EVSZ visszaállítása az mpx tárba
Uzemmód
Byte
Burst
FK beállítása 0-ra
Visszatoltés
A CPU program folytatása
Utasításlehivás?
Nem
Igen
A maszkengedélyezés-indikátor beállítva?
Nem
Igen
CSW kialakítása
CSW tárolása az FT-ben
CS-VE cim tárolása a B/K régi PSW -ben
A pillanatnyi PSW tárolása B/K régi PSW-ként
Az FT-ben levő új B/K PSW átvitel a pillanatnyi PSW helyre a helyi tárban
A megszakítási program végrehajtása

10.21/a ábra

Itt nincs szükség visszamozgatásra. Az átvitel addig folytatódik, amíg csak csatornavég jelet nem érzékel a csatorna.

Ekkor, akár byte-, akár burst-üzemmódban van a csatorna, megvizsgálják a csatornaparancsszóban a láncolás jelzőbiteket, a láncolás meghatározására. Ha van láncolás, a csatorna, a vezérlőegységtől jövő készülékvég jel fogadása után lehívja a következő csatornaparancsszót és végrehajtja a parancsot. Ez a folyamat addig folytatódik, amíg a SIO összes csatornaparancsszava végre nem hajtódik. Ezután megszakítást kezdeményeznek, és a csatorna készen áll a működés befejezésére.

A csatorna akkor fogadja el a megszakítást, ha nem várakozik korábbi megszakításkiszolgálásra. Ekkor megvizsgálják a rendszermaszkot, annak meghatározására, hogy megengedhető-e ez a megszakítás a csatornán. Ha a maszkolt csatorna megengedhető, maszkolható megszakításbitnek adnak értéket. Ha a csatorna nem fogadja el a megszakítást, a csatorna kéri a vezérlőegységet, rakja le a vezérlőegységben levő státusz byte-ot. Akár elfogadja a csatorna a megszakítást, akár nem, visszaállítja az egységvezérlő szót az mpx tárba.

Ha a csatorna byte-üzemmódban van, visszamozgatás kerül végrehajtásra; ha burst-üzemmódban van, a feltételkód 0 értéket kap. Bármelyik esetben, a B/K megszakításfogadás előtt folyamatban levő CPU program folytatódik. Ha a CPU éppen nem hív le egy utasítást, vagy ha megteszi, de a maszkengedélyezés-indikátor nincs beállítva, a CPU folytatja a program végrehajtását. Ha a központi egység egy utasítást hív le, és a maszkengedélyezés-indikátor be van állítva, a B/K megszakítás-mikroprogram lép be, és csatornaállapot-szó (CSSZ) kerül kialakításra. Ez a CSSZ határozza meg a csatorna és a készülék állapotát az éppen végrehajtott be-, ill. kimeneti művelet befejeztekor, és a főtárban tárolódik. Továbbá, a csatornaegység címe lesz a pillanatnyi programállapotszó (PSW) megszakításkódja, és a pillanatnyi PSW-t a B/K régi PSW-jeként tárolja. Ez a programállapotszó határozza meg azt a csatornát és egységét, amely a megszakítást okozta, míg a csatornaállapot-szóból állapítható meg, hogy miért jött létre a megszakítás. Az új B/K programállapotszót lehívják a főtárból, és betöltik a pillanatnyi PSW helyére a helyi tárolóba. Ekkor befejeződik a B/K megszakítás automatikus feldolgozása a mikroprogrammal, és az új PSW utasítás számlálója megadja egy megszakítási szubrutin kezdőcímét.

Irodalom

[1] ECKERT, J. P., JR.—WEINER, J. R.—WLSH, H. F.—MITCHELL, H. F.: *The UNIVAC System.* Review of Electronic Digital Computers, Joint AIEE-IRE Computer Conference, February, 1952.

[2] BUCHHOLZ, W.: *The System Design of the IBM Type 701 Computer.* Proceedings of the IRE, October, 1953, pp. 1262...1275.

[3] FORGIE, J. W.: *The Lincoln TX-2 Input-Output System.* Proceedings of J. W. C. C., 1957, pp. 156...160.

[4] BUCHHOLZ, W., ed. H. H.: *Planning a Computer System.* New York, McGraw-Hill Book Company, 1962.

[5] FAGG, P.—BROWN, J. L.—HIPP, J. A.—DOODY, D. T.: *IBM System/360 Engineering.* Proceedings of Fall Joint Computer Conference, 1964, pp. 205...231.

[6] PADEGS, A.: *The Structure of System/360, Part IV, Channel Design Considerations.* IBM Systems Journal, 3, Nos. 2 and 3, 1964, pp. 165...180.

[7] BLAAUW, G. A.: *The Structure of System/360, Part V, Multisystem Organization.* IBM Systems Journal, 3, Nos. 2 and 3, 1964. pp. 181...195.

[8] HASSITT, A.: *Computer Programming and Computer Systems.* New York; Academic Press Inc., 1967.

[9] GSCHWIND, H. W.: *Desing of Digital Computers.* Springer-Verlag New York Inc., 1967.

[10] CHU W.W.: *A Study of Synchronous Time Division Multiplexing for Time-sharing Computer Systems.* Proceedings of the FJCC, 1969.

[11] PHORNTON, J. E.: *Design of a Computer: the Control Data 6600.* Glenview, Illinois, Scott, Foresman and Company, 1970.

11.
A software mikroprogramozott implementálása

Az előző fejezetekben bemutattuk, hogy a hardware mikroprogramozással implementálható. A hardware mikroprogramozott implementálásának több előnye van, pl. az ilyen implementálás rugalmasan változtatható és a mikroprogram-tervezés hasonló a software-tervezéshez. Az implementálás rugalmassága — bizonyos határok között — lehetővé teszi, hogy a hardware változtatása nélkül különböző mikroprogramokat használjunk különböző algoritmusok realizálására abban az esetben, ha a mikroprogramtár változtatható ROM tár vagy írható—olvasható tár. A software-hasonlóság pedig lehetőséget ad a programozási módszerek alkalmazására a mikroprogramozásban.

Az előző fejezetekben azt is megmutattuk, hogy a hardware éppúgy algoritmusokat implementál, mint a software. Ezért a software is implementálható mikroprogramozással. A software-nek mikroprogramozással történő implementálásából két előny származik: a megvalósítás rugalmasságának megőrzése és a végrehajtási sebesség növelése. Ebben a fejezetben példán mutatjuk be a software mikroprogramozásának lehetőségét. A példában részletesen ismertetjük áthelyezhető kódból végrehajtható kódra fordítás mikroprogramozott megvalósítását, ez a fordítás a programtöltőnek, vagy röviden betöltőnek nevezett software egyik része.

11.1.
Áthelyezhető kód végrehajtható kóddá való fordítása

A számítógéprendszer programokat hajt végre. Az egyes kész programokat gyakran *munkának* (job-nak) hívják. Adagolt (batch) üzemű számítógéprendszerben a munkákat egy adagba (batch) egyesítik, és azután egymás után dolgoztatják fel őket a számítógéprendszerrel, egészen addig, amíg az adag nem ürül ki. Az egyes munkák rendszerint több részprogramból állnak. Részprogram lehet egy főprogram, szubrutin vagy független kódrészlet. Ezek a részprogramok egy vagy több forrásnyelven fordulhatnak elő, amely pl. compiler nyelv, assembly nyelv vagy bináris kód lehet. Annak érdekében, hogy egy munka teljes programját végre tudja a számítógép hajtani, ezt egyet-

len abszolút című integrált programmá kell összerakni, és így kell betölteni a főtárba. Az ezt eredményező folyamatot három fázisra lehet osztani:

— összeszerkesztő fázisra, amelyben minden egyes részprogramot *áthelyezhető kóddá* (relocatable) fordítanak le; ekkor a munka áthelyezhető kódok összességéből áll;

— összekapcsolási (linkage editing) fázisra, amelyben ezeket az áthelyezhető kódokat egyetlen áthelyezhető kóddá kapcsolják össze;

— betöltési (loading) fázisra, amelyben az áthelyezhető kód *végrehajtható kódra* fordítódik le, és a végrehajtható kód a főtárba kerül. A software mikroprogramozott megvalósításának illusztrálására leírjuk az áthelyezhető kódnak hardware-rel végzett fordítását végrehajtható kóddá. Az egyszerűség kedvéért a *betöltő* elnevezést fogjuk használni ennek a fordításnak jelölésére.

Minthogy a legtöbb számítógépben viszonylag kicsi a főtár kapacitása, a főtár kapacitásánál nagyobb programot (vagyis, annak végrehajtható kódját) nem lehet teljes egészében betölteni a főtárba. Megszokott eljárás a felülírás (overlay) alkalmazása. *Felülírás* alatt azt értjük, hogy a programozó programját több, független tárbetöltésre osztja fel (amelyek közül mindegyiknek kisebbnek kell lennie, mint a tár kapacitása) speciális vezérlő kártyák felhasználásával. A pillanatnyilag aktív tárbetöltés behívhat egy másik mágnesmagos tárbetöltést, saját maga felülírására. Egy tárterületet megőriznek a felülírástól, így ez a terület tartalmazhat olyan információkat, amelyeket tárbetöltéstől tárbetöltésig tovább lehet adni. Az egyszerűség kedvéért ismertetésünk során feltételezzük, hogy a most bemutatásra kerülő betöltőben nincs szükség átírási lehetőségre.

11.1.1. Áthelyezhető elemek

A betöltő bemenete egy vagy több áthelyezhető elem. Mindegyik *áthelyezhető elem* két részből áll:

a) az áthelyezhető kódból és
b) a szimbolikus címtáblázatokból.

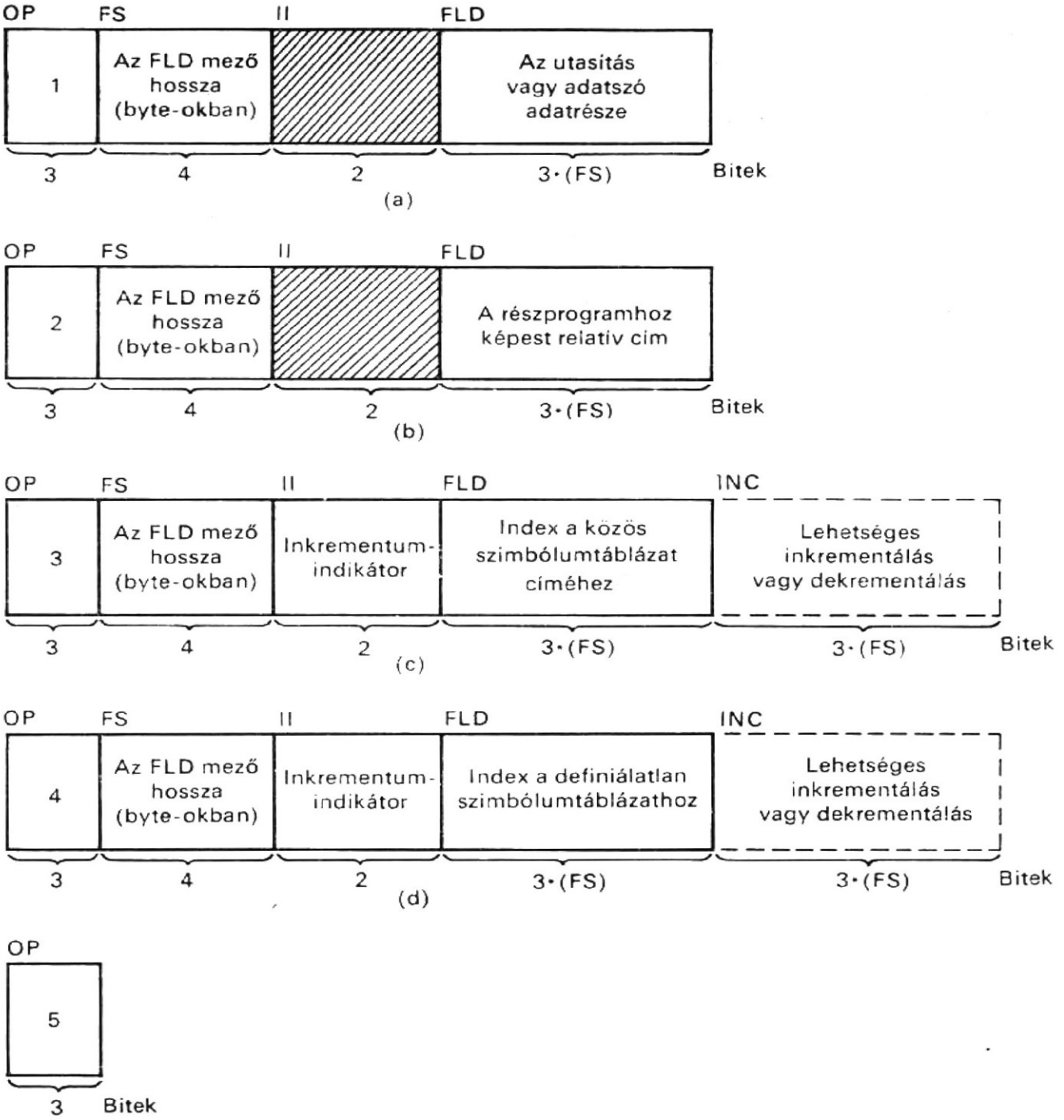
Az *áthelyezhető kód* a gépi nyelv utasításainak és adatainak közbenső formája, amely assembly nyelvű részprogram összeszerkesztéséből jön létre. A gépi nyelvű utasítás rendszerint egy nem címtípusú részből (pl. a műveledi kód részből), és egy cím részből tevődik össze. Minthogy az áthelyezhető fordítás csak a címek összeillesztését igényli, ezért csak az utasítás cím része és nem cím része közötti megkülönböztetésre van szükség. Ezért az áthelyezhető kódot *áthelyezhető szavak* sorozatának tekinthetjük. Mindegyik áthelyezhető szó egy utasításnak címrészét vagy egy nem cím részét tartalmazza. Ilyen összefüggésben az adatszót cím rész nélküli utasításnak tekinthetjük.

A választott példa áthelyezhető szóformátumai a 11.1. ábrán láthatók. Az egyes formátumoknak egytől öt mezőjük lehet:

OP, FS, II, FLD és INC

Az FLD mező tartalmazza az adatokat vagy egy utasítás címmezejét, vagy indexet valamelyik táblázathoz. Az FS mező az FLD mező hosszát jelöli, oktális számrendszerben. Az OP mező a következőképpen identifikálja az FLD mezőt:

1. Adat (OP=1).
2. Relatív cím (OP=2). Ez a cím a részprogramon belüli, a *részprogram címéhez* képest relatív helyre hivatkozik. A részprogram címe, ahová a végrehajtható kódot be kell tölteni.



11.1. ábra
Áthelyezhető szóformátumok
***a)* A formátum, adatot jelöl; *b)* B formátum, relatív címet jelöl; *c)* C formátum, közös adatcímet jelöl; *d)* D formátum, külső címet jelöl; *e)* E formátum, áthelyezhető kód végét jelöli**

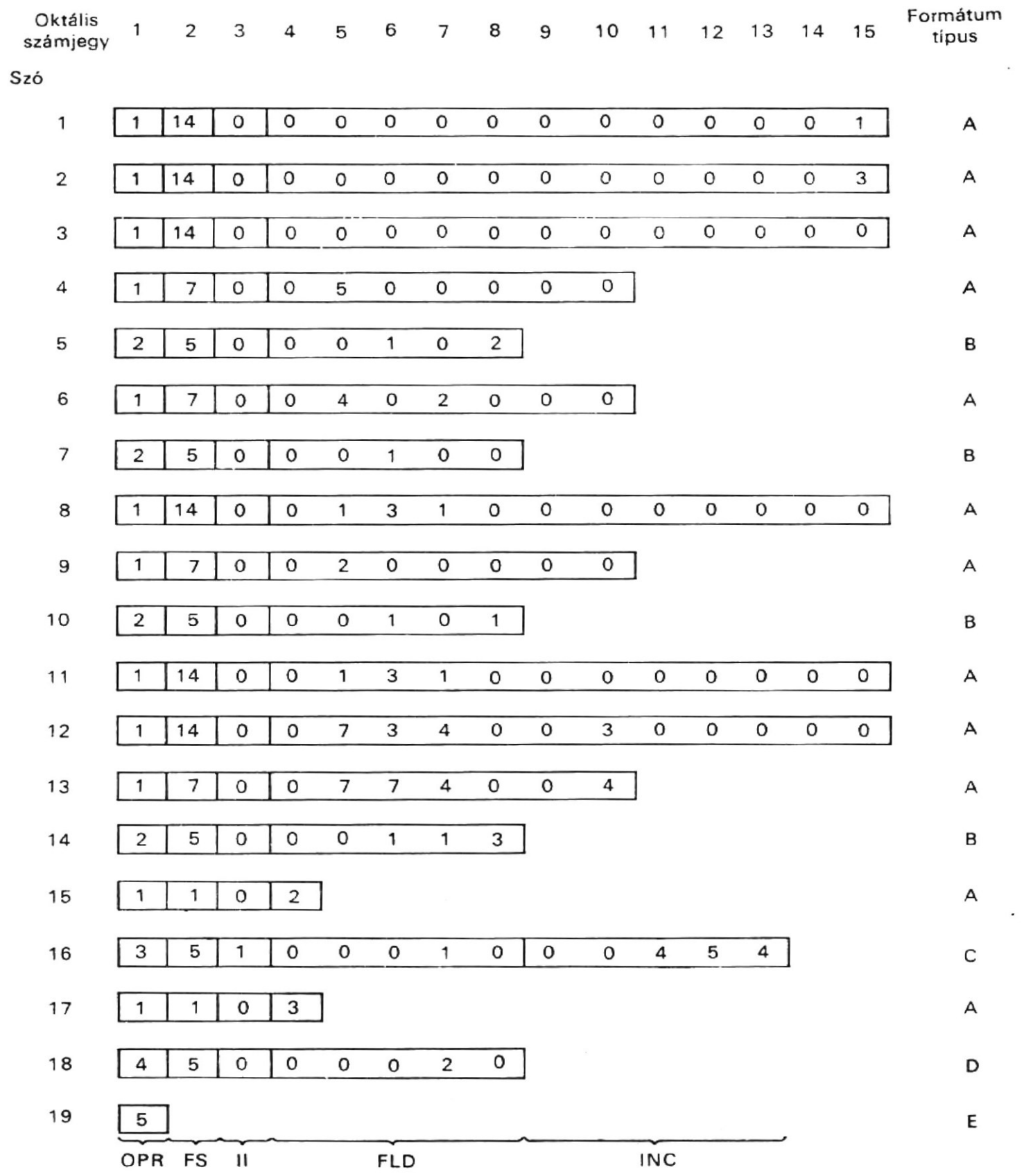
3. Közös adatok címe (OP=3). Ez egy külső cím, amely olyan adatterületre hivatkozik, amely több részprogramra nézve közös.

4. Külső cím (OP=4). Ez olyan külső cím, amely más részprogramokba való belépési pontra hivatkozik.

5. Az áthelyezhető kód vége (OP=5).

Az II és INC mezők numerikus növekményt adnak ezekhez a szimbolikus címekhez. Ha II 0-val egyenlő, nincs INC mező. Ha II 1-gyel, ill. 2-vel egyenlő, a címet növelik, ill. csökkentik az INC mező tartalmával.

A 11.2. ábrán áthelyezhető kódra láthatunk példát, ahol a mezőket függőleges vonalak választják el egymástól, és a számok oktálisak. Az 1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 15 és 17 szavak adatokkal rendelkező áthelyezhető szavak (OP=1). Az 5, 7, 10 és 14 szó relatív című szó (OP=2). A 16. szó közös adatcímű szó (OP=3). A 18. szó külső



11.2. ábra
Példa áthelyezhető kódra (oktális számjegyekkel)

címmel ellátott szó (OP=4). A 19. szó az áthelyezhető kód végét jelző szó (OP=5). Ezeknek az áthelyezhető szavaknak különböző a hosszuk. Bár balra illeszkedve mutatjuk be őket, valójában a tárban tárolt bitek láncolatai (11.3. ábra).

Az áthelyezhető elem relatív címe, közös adatcíme és külső címe mind szimbolikus. Ezeket arra használják, hogy a részprogramokat összekapcsolják egymással. A következő három *szimbolikus címtáblázatban* tárolják őket:

1	6	0	0	0	0	0	0	0	0	0	0	1
0	0	1	1	6	0	0	0	0	0	0	0	2
0	0	0	0	0	3	1	6	0	0	0	0	3
0	0	0	0	0	0	0	0	0	1	3	4	4
0	5	0	0	0	0	0	2	2	4	0	0	5
1	0	2	1	3	4	0	4	0	2	0	0	6
0	2	2	4	0	0	1	0	0	1	6	0	7
0	1	3	1	0	0	0	0	0	0	0	0	8
1	3	4	0	2	0	0	0	0	0	2	2	9
4	0	0	1	0	1	1	6	0	0	1	3	10
1	0	0	0	0	0	0	0	0	1	6	0	11
0	7	3	4	0	0	3	0	0	0	0	0	12
1	3	4	0	7	7	4	0	0	4	2	2	13
4	0	0	1	1	3	1	0	4	2	3	2	14
5	0	0	0	1	0	0	0	4	5	4	1	15
0	4	3	4	2	4	0	0	0	2	0	5	16

Egy szó a tárban
(36 bit)

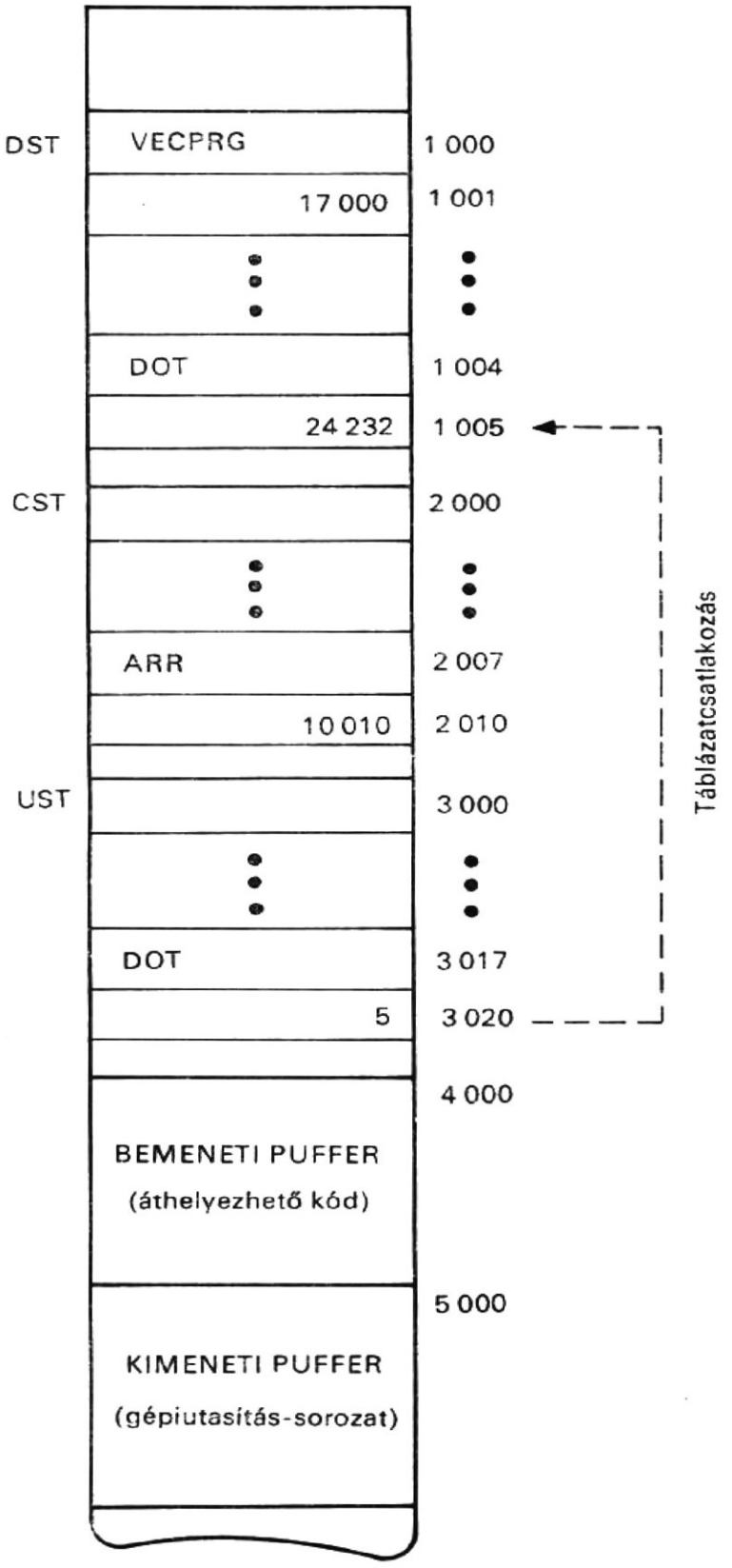
11.3. ábra
Példa áthelyezhető kód elhelyezésére a tárban (a kettőzött vonalak áthelyezhető szavakat választanak el egymástól; valamennyi számjegy oktális)

1. Definiált szimbólumtáblázat (DST). Ez a táblázat tartalmazza a részprogramban mindegyik belépési pont szimbolikus nevét és relatív címét.

2. Definiálatlan szimbólumtáblázat (UST). Ez a táblázat a részprogramban levő, egyes külső címek (OP=4) szimbolikus nevét tartalmazza. Az az utasítás a részprogramban, amely a külső címre hivatkozik, tartalmaz egy cím-csatlakozást ennek a külső címnek a belépési pontjához az UST-táblázatban.

3. Közös szimbólumtáblázat (CST). Ez a táblázat tartalmazza a részprogramban levő minden egyes közös adat címének (OP=3) szimbolikus nevét. A részprogramban az az utasítás, amelyik a közös adatcímre hivatkozik, tartalmaz egy címcsatlakozást a közös adatcímnek belépési pontjára vonatkozóan a CST-táblázatban.

A 11.4. ábrán ezt a három táblázatot ábrázoltuk abban a formában, ahogy a főtárban megjelenik; az ábrára a továbbiakban még hivatkozni fogunk.



11.4. ábra
Példa táblázatokra és pufferekre a főtárban

11.1.2. Végrehajtható kód

A tárolt programú számítógépet abszolút című (vagyis főtár című) gépi utasítások végrehajtására tervezik. Az ilyen, gépi utasításokban megírt programokat *végrehajtható kódú* programoknak nevezik. A betöltőtől jövő kód egy végrehajtható kód. A végrehajtható kód első szava tárolhatja azt az abszolút címet, amelynél a végrehajtható kódnak a főtárba történő betöltése elkezdődik.

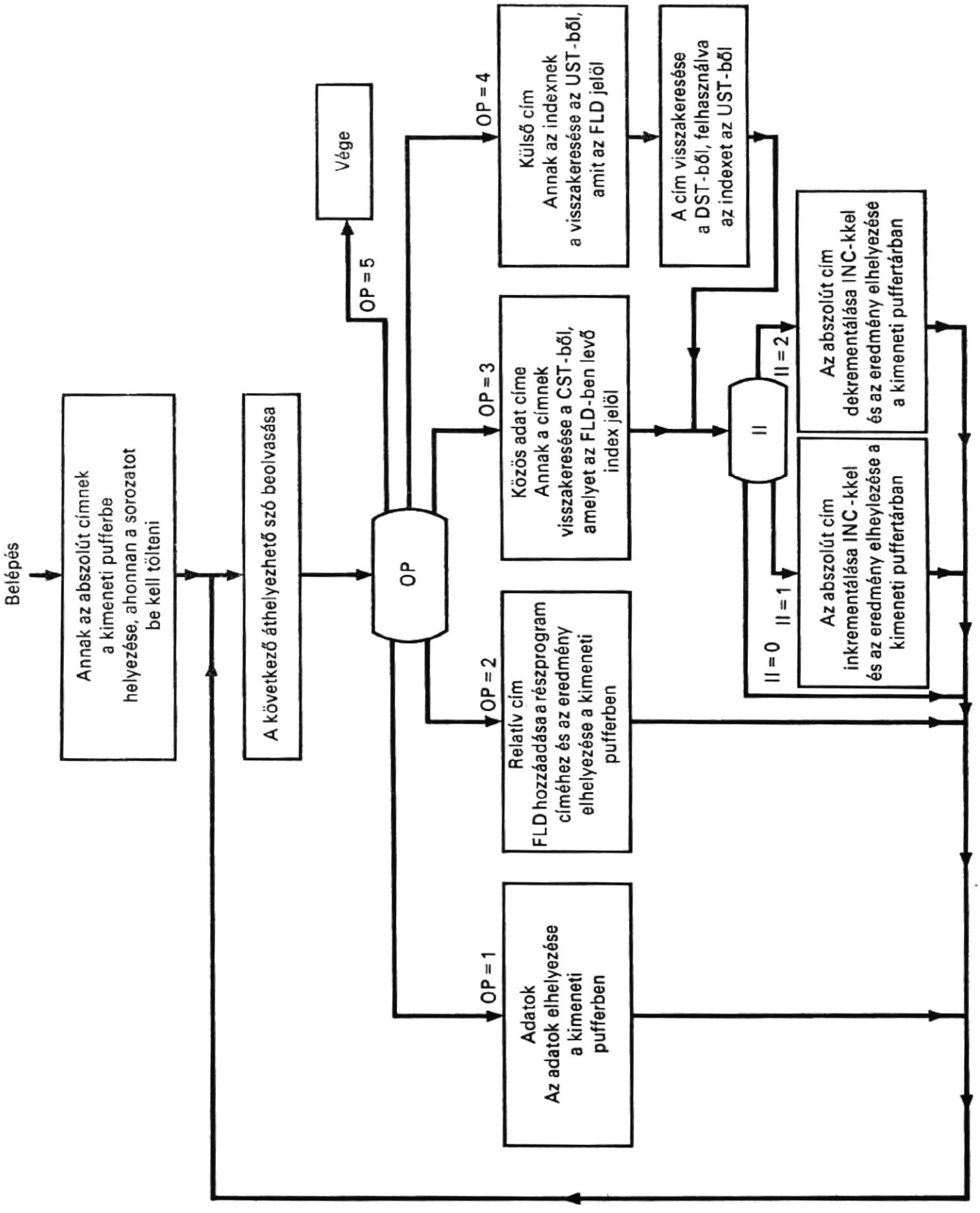
1	0	0	0	0	0	0	0	1	7	0	0	0
						⋮						
65	0	0	0	0	0	0	0	0	0	0	0	1
66	0	0	0	0	0	0	0	0	0	0	0	3
67	0	0	0	0	0	0	0	0	0	0	0	0
68	0	5	0	0	0	0	0	1	7	1	0	2
69	0	4	0	2	0	0	0	1	7	1	0	0
70	0	1	3	1	0	0	0	0	0	0	0	0
71	0	2	0	0	0	0	0	1	7	1	0	1
72	0	1	3	1	0	0	0	0	0	0	0	0
73	0	7	3	4	0	0	3	0	0	0	0	0
74	0	7	7	4	0	0	4	1	7	1	1	3
75	2	1	0	4	6	4	3	2	4	2	3	2

11.5. ábra
Végrehajtható kód a főtárban

A 11.5. ábra végrehajtható kódra mutat példát; ez a 11.2. ábrán látható áthelyezhető kód fordításából származhat. A 11.5. ábrán az utasítás- és adatformátumok az IBM 7090 számítógépcsaládjában használt formátumokat követik. Amint látható, az első szó a 17000_8 abszolút címet tartalmazza. A 65...75. helyeket elfoglaló végrehajtható kódról feltételezzük, hogy egy nagyobb program része. A betöltő a 19 áthelyezhető szót 12 gépi utasításra fordítja le, mivel a gépi utasítások közül néhány olyan címrészeket és adatrészeket tartalmaz, amelyeket egynél több áthelyezhető szó ír le. A 11.2. ábrán látható 1., 2., 3., 8., 11. és 12. szó FLD mezőiből a 11.5. ábrán rendre, a 65., 66., 67., 70., 72. és 73. szó lesz, mivel ezeknek az áthelyezhető szavaknak nincs címrészük.

11.1.3. Fordítási algoritmus

A fordítási folyamat először lazítja (unpack) a bemeneti pufferben tárolt áthelyezhető kódot, azután értelmezi az áthelyezhető kód egyes szavainak műveleti mezejét, végül az adatokat vagy a módosított címet gépi nyelvű utasítások sorozatává alakítja át (szerkeszti össze) a kimeneti pufferben. A pufferek a 11.4. ábrán láthatók. Magát a fordítási folyamatot a 11.6. folyamatábra szemlélteti. Amint az ábrán látható, a kezdőlépés elhelyezi a részprogram címét a kimeneti puffer területén. Kiolvassák szétválasztás céljából a bemeneti puffer területről az első vagy a legközelebbi áthelyezhető szót. A szétválasztási folyamat ismeri föl az áthelyezhető szó határait, valamint az egyes szavak mezőinek határait. Először dekódolják az OP mezőt, majd a címet az alábbiak szerint módosítják.



11.6. ábra
A fordítási algoritmus folyamatábrája

1. Ha az OP=1, ez adatot jelez (a 11.1. ábrán látható A formátum). Minthogy ez nem cím, nincs szükség címmódosításra. Az FLD mező FS oktális számjegyeit elhelyezik a kimeneti pufferterület legközelebb rendelkezésre álló oktális számjegypozícióin, ahol az FS az FLD mező méretét jelöli, oktális számrendszerben.

2. Ha az OP=2, ez relatív címet jelöl (B formátum). Az FLD mező FS oktális számjegyeit (a relatív címet) hozzáadják a részprogram címéhez (mint pl. a 17000_8 a 11.5. ábrán). Az összeget a kimeneti pufferterület legközelebbi FS oktális számjegypozícióin helyezik el.

3. Ha az OP=3, közös adat címét jelöli (C formátum). Az FLD mező FS oktális számjegyei az indexek. Az index egy, a közös szimbólumtáblázat (CST) helyéhez viszonyított cím (mint pl. a 2000_8 a 11.4. ábrán). Az ezen a helyen tárolt abszolút címet keresik vissza. Ha az II mező 0-t tartalmaz, nincs címmódosítás. Ha az II mező 1-et, ill. 2-t tartalmaz, az INC tartalmát hozzáadják, ill. kivonják az abszolút címhez(ből). A három eset bármelyikében azután az eredményképpen kapott címet a kimeneti pufferterület legközelebbi FS oktális számjegypozícióin helyezik el.

4. Ha az OP=4, külső címet jelez (D formátum). Az FLD mező FS oktális számjegyeit (amely az UST-hez viszonyított relatív cím) hozzáadják a definiálatlan szimbólumtáblázat (UST) címéhez (mint pl. a 3000_8 a 11.4. ábrán). Az ezen a helyen tárolt indexet visszakeresik és hozzáadják a definiált szimbólumtáblázat (DST) címéhez (mint pl. a 1000_8 a 11.4. ábrán). Ezután visszakeresik ezen a helyen az abszolút címet. Ha az II mező 0, nincs szükség további címmódosításra. Ha az II mező 1, ill. 2, az INC mező tartalmát hozzáadják, ill. levonják az abszolút címhez(ből). Mindhárom esetben az abszolút címet a kimeneti pufferterület legközelebbi FS oktális számjegypozícióin helyezik el.

5. Ha az OP=5, az áthelyezhető kód végét jelöli (E formátum). A fordítás ezzel befejeződött. Miután végrehajtódott a fenti művelet, a következő áthelyezhető szót olvassák ki a bemeneti pufferterületről. Folytatódik a szétválasztás, a dekódolás és a címmódosítás, egészen addig, amíg el nem érik az áthelyezhető kód végét. Ezzel a fordítás készen van. Ezt a fordítási folyamatot fogjuk a későbbiekben részletesebben leírni, amikor majd meghatározzuk a folyamatábrákat.

Példaképpen, legyen a 11.2. és 11.3. ábrán levő áthelyezhető kód az a bemenet, amelyet a 11.4. ábrán látható tárbemeneti puffer területén tárolnak. Legyen a 11.5. ábra végrehajtható kódja a fordításból kapott kimenet, amelyet az ugyancsak a 11.4. ábrán látható tárkimeneti puffer területén tárolnak. A 11.5. ábrán az 1. szó tartalmazza a 17000_8 abszolút címet, amelytől kezdve a soron következő gépi nyelvű utasítássorozatot töltik be. Feltételezzük, hogy a 2...64 című szavak a részprogramnak valamilyen más részét képezik. A 11.2. ábra 1., 2. és 3. áthelyezhető szavai tartalmazzák az adatokat, és így ezeket módosítás nélkül fordítják le a 11.4. ábra kimeneti pufferterületének 65., 66. és 67. szavaivá. A 11.2. ábrán a 4. áthelyezhető szó ugyancsak adatot tartalmaz, és módosítás nélkül fordítódik le a 11.5. ábra 68. szavának első hét oktális számjegyévé. A 11.2. ábra 5. szava relatív címet tartalmaz; tehát először hozzáadják az FLD mező tartalmát a részprogram címéhez (17000_8) és azután az eredményt elhelyezik a 68. szó utolsó öt oktális számjegyében (17102_8). A 11.5. ábra 69....74. szavait hasonlóképpen fordítják le a 11.2. ábra 6...14. áthelyezhető szavaiból. A 11.5 ábrán látható 75. szó kétcímes gépi utasítás; ezt a 11.2. ábra 15...18. áthelyezhető szavaiból fordították le. A 15. áthelyezhető szó, amely az utasítás op-kódját tartalmazza, lesz a 75. szó első nyolc oktális számjegye. A 16. áthelyezhető szó a közös szimbólumtáblázatra (CST) indexet tartalmaz (00010_8). A 11.4. ábrán a CST címéhez képest nyolcadik helyen (2000_8) az található, hogy az abszolút cím 10010_8, és ennek szimbolikus neve ARR. Minthogy az II mező 1, a 10010_8 címet megnövelik az INC mező tartalmával; az eredmény 10464_8, és a 75. szó 2...6 oktális számjegyeit foglalja el. A 17. áthelyezhető szó, amely az utasítás indexét tartalmazza, módosítás nélkül kerül

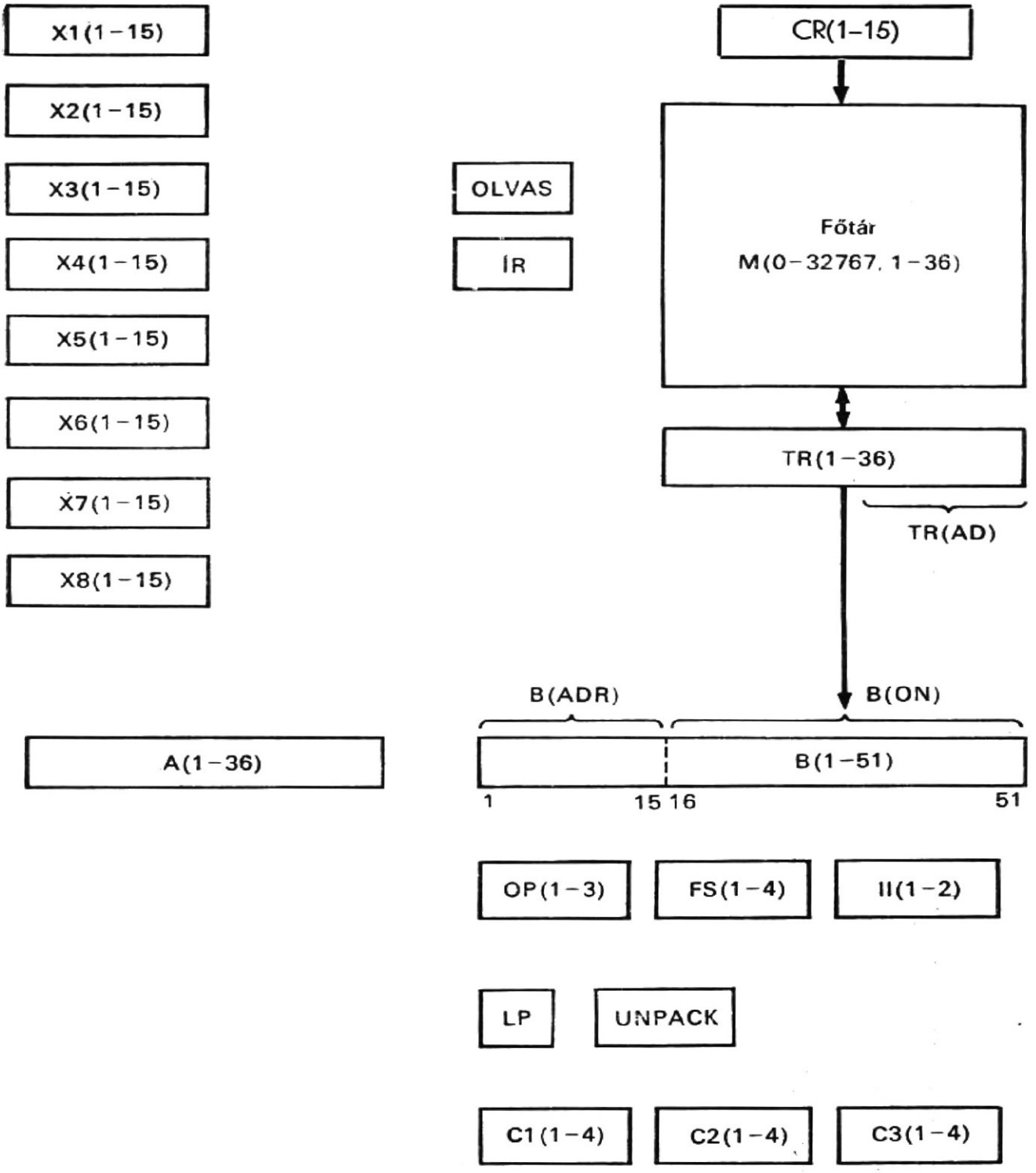
lefordításra a 75. szó 7. oktális számjegyébe. A 18. áthelyezhető szó indexet tartalmaz (00020_8) a definiálatlan szimbólumtáblázathoz (UST) számára. Az UST címhez képest tizenhatodik helyen (3000_8) található meg a külső cím, amely 5 és amelynek szimbolikus neve DOT (11.4. ábra). Ez a cím (005_8) a definiált szimbólumtáblázat, a DST helyéhez (1000_8) képest relatív. Ezen a helyen (1005_8) a 24232_8 abszolút cím található. Minthogy a 18. áthelyezhető szó II mezeje 0-t tartalmaz, nincs szükség címmódosításra; ez az abszolút cím a DOT belépési pont. A 19. áthelyezhető szó jelzi az áthelyezhető kód végét.

11.2. Struktúra

A 11.7. ábrán láthatjuk a mikroprogramozott betöltő struktúráját, ez nem tartalmazza a vezérlési részt, amelyet később fogunk bemutatni. Az M főtárnak van egy CR címregisztere és egy TR tartalomregisztere. Az egybites OLVAS és IR regisztereket a tárból történő olvasás, ill. tárba történő írás vezérlésére használják. Az áthelyezhető elemeket és a bemeneti és kimeneti pufferterületeket a főtár tárolja. Nyolc indexregisztert, X1, X2, ..., X8 alkalmazunk, amelyek a táblát és a puffercímeket tárolják a fordítás alatt. Az OP, FS, ill. II regiszterek rendre az áthelyezhető szó OP-mezejét, FS-mezejét, ill. II-mezejét tárolják. Az áthelyezhető szó szétválasztását és cím részének címmódosítását az A és B regiszterekben végzik. Az egybites LP regiszter jelzi, hogy a B regisztert, vagy az A-B regisztersort balra léptették az LP regiszter szerint, amely a szétválasztási sorozat hívására szolgál. Ehhez járul még három számláló, a C1, C2 és C3. Ezt a struktúrát az alábbiakban írjuk le.

Comment, a fordító struktúrája (11.1)

Register,	A(1—36),	$akkumulátor
	B(1—51),	$szétválasztás-regiszter
	CR(1—15),	$címregiszter
	TR(1—36),	$tartalomregiszter
	X1(1—15),	$az INC mezőt tárolja
	X2(1—15),	$a BEMENET mezőt tárolja
	X3(1—15),	$a KIMENET mezőt tárolja
	X4(1—15),	$a CST-t tárolja
	X5(1—15),	$az UST-t tárolja
	X6(1—15),	$a DST-t tárolja
	X7(1—15),	$a részprogram címét tárolja
	X8(1—15),	$átmeneti tár
	OP(1—3),	$op-regiszter
	FS(1—4),	$mezőméret-regiszter
	II(1—2),	$inkrementálás-indikátor
	C1(1—4),	$balra történő léptetéseket számlálja a regisztersorban
	C2(1—4),	$balra történő léptetéseket számlálja a B regiszterben
	C3(1—4),	$balra történő léptetéseket számlálja az A regiszterben
	LP,	$léptetésvezérlő regiszter
	UNPACK,	$vezérlőregiszter



11.7. ábra
Betöltő struktúrája

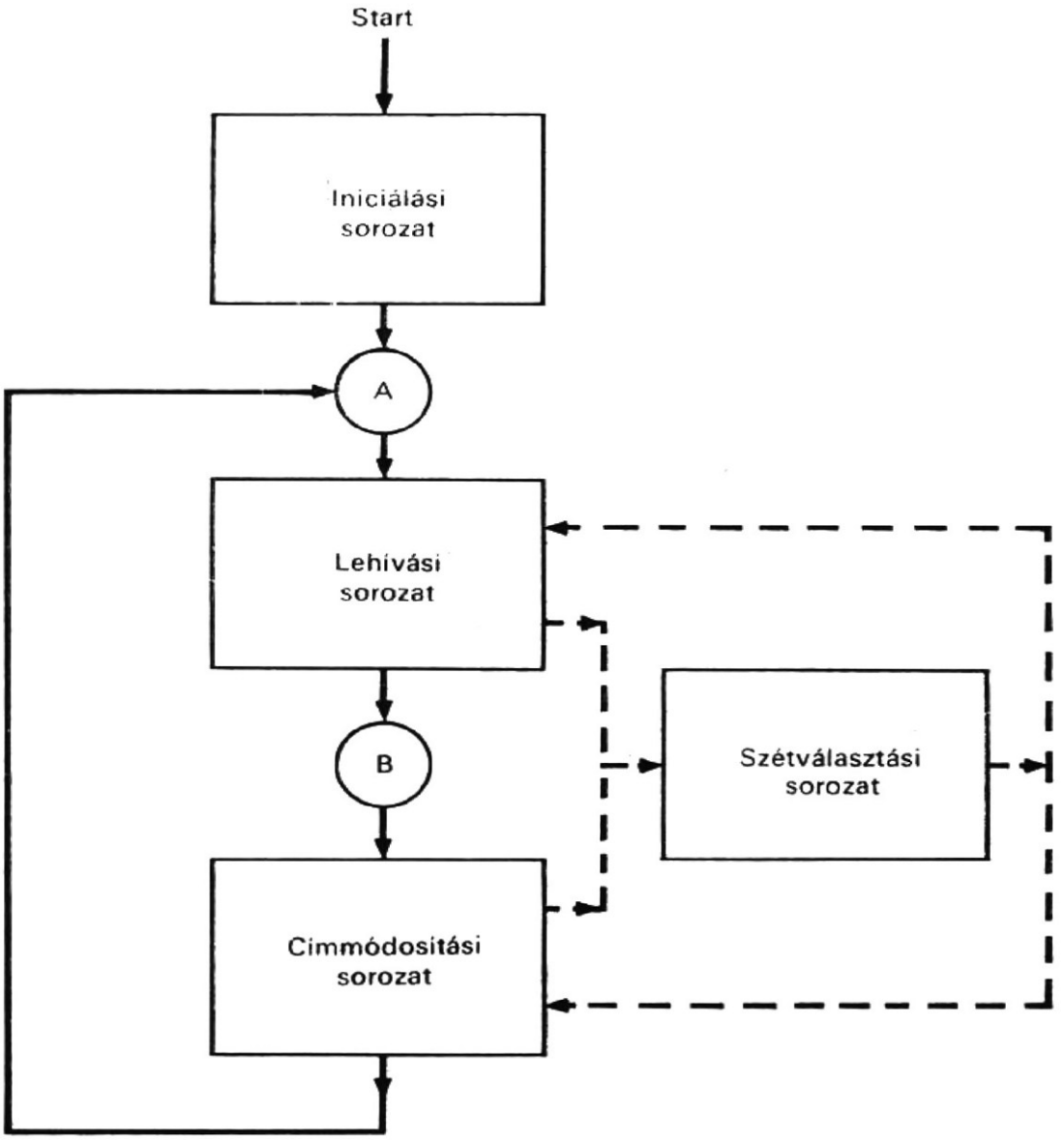
```
                OLVAS,                  $tárolóból olvasás vezérlőregiszter
                ÍR,                     $tárolóba írás vezérlőregiszter
Subregister,    B(ADR)=B(1—15),         $a szétválasztásregiszter címrésze
                B(ON)=B(16—51),         $a szétválasztásregiszter bemeneti
                                          része
                TR(AD)=TR(22—36),       $a tartalomregiszter címrésze
Memory,         M(CR)=M(0—32767,1—36),  $főtár
Casregister,    AB(1—87)=A—B,
```

11.3. Műveletsorozatok

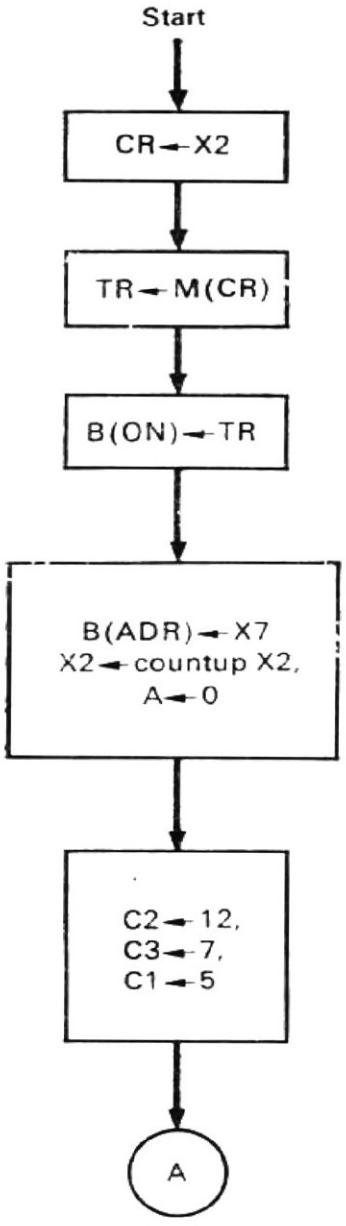
A 11.6. ábrán bemutatott fordítási algoritmust most folyamatábrákra bontjuk szét. A 11.8. folyamatábra a fordító négy műveletsorozatát tartalmazza:

— iniciálás,
— lehívás,
— címmódosítás és
— szétválasztás.

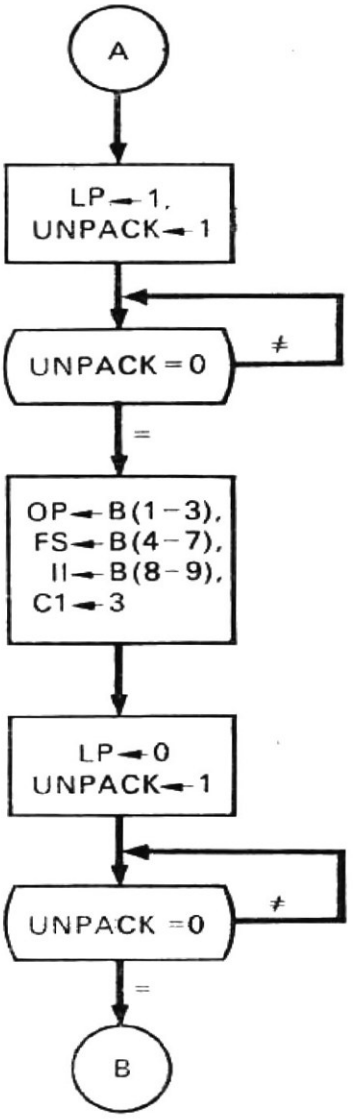
E négy műveletsorozat részletes folyamatábráit a 11.9...11.12. ábrákon láthatjuk. Az iniciáló műveletsorozat kezdeményezi a fordítást. Az iniciáló műveletsorozat hívja le az áthelyezhető szót, szétválasztja és dekódolja azt. A címmódosítási sorozat hajtja végre a címmódosítást. A szétválasztási műveletsorozat végzi el az áthelyezhető szó kiolvasását a bemeneti pufferterületről, az AB regisztersor balra léptetését és a gépi utasítás elhelyezését a kimeneti pufferterületen. Amint a szaggatott vonallal jelöltük, a szétválasztási sorozatot a lehívási sorozat és a címmódosítási sorozat alatt hívják be.



11.8. ábra
A betöltő négy sorozata



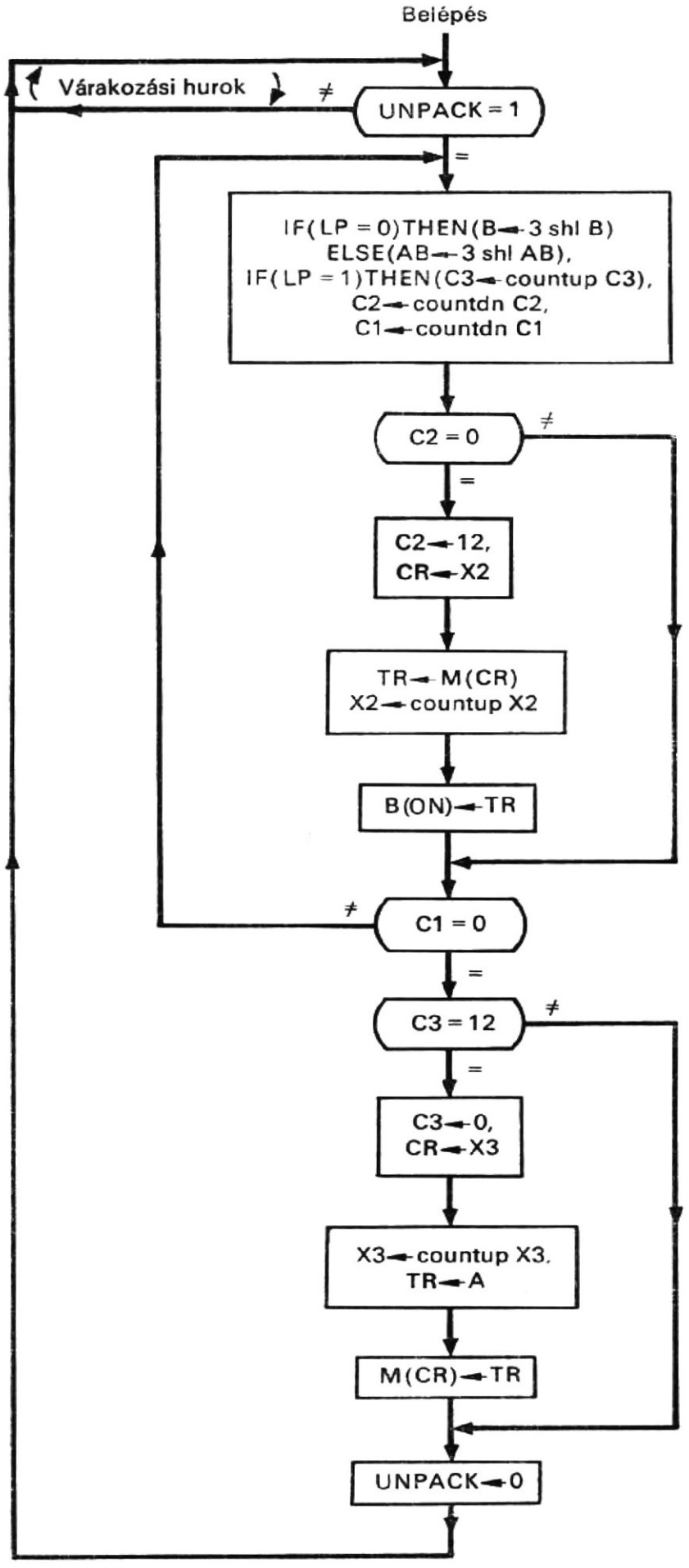
11.9. ábra
Az indítási sorozat folyamatábrája



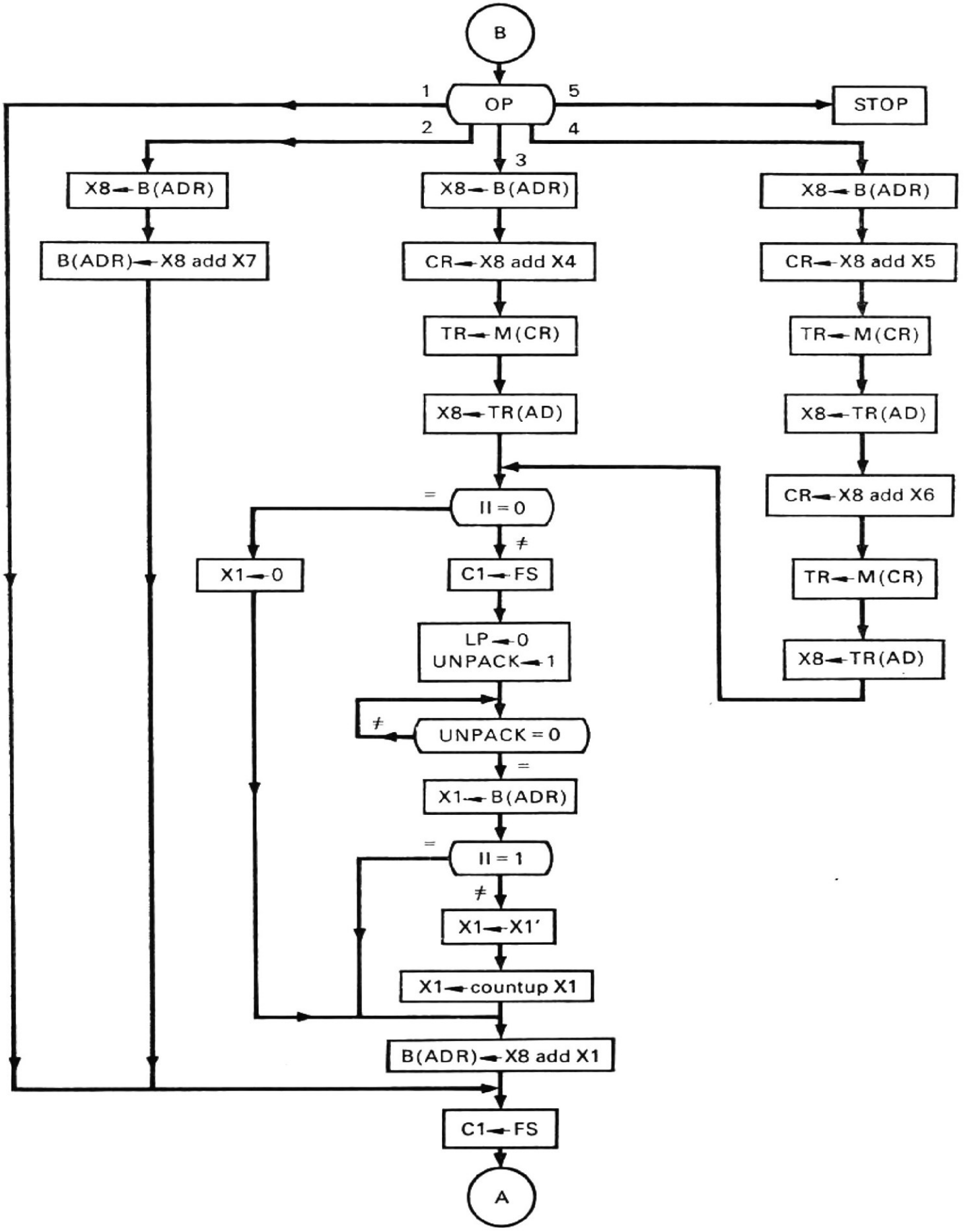
11.10. ábra
A lehívási sorozat folyamatábrája

Tételezzük föl, hogy az áthelyezhető elem és a pufferek kezdetben a tárban vannak, és hogy az áthelyezhető elem bitek sorozata (string), amint a 11.3. ábrán látható. Feltesszük, hogy a táblázatok és a pufferterületek címei az indexregiszterekben vannak az alábbiak szerint:

1. A bemeneti puffer címe az X2 regiszterben,
2. A kimeneti puffer címe az X3 regiszterben,
3. A CST címe az X4 regiszterben,
4. Az UST címe az X5 regiszterben,
5. A DST címe az X6 regiszterben,
6. A részprogram címe az X7 regiszterben.



11.11. ábra
A szétválogatási részsorozat folyamatábrája



11.12. ábra
A címmódosítási sorozat

11.3.1. Iniciálási sorozat

A 11.9. ábrán látható iniciálási sorozat öt feladatot old meg. A pufferterületről (amelynek címe az X2 regiszterben van) az első s a B(ON) részregiszterben. A részprogramnak az X7 regiszterben le a B(ADR) részregiszterbe. Az X2 regiszterértékét 1-gyel megnöveli. A 0 értéket ad. A C1, C2, ill. C3 számlálók kezdeti értékét rendre 5-re állítja be.

A C1 számláló számlálja az AB regisztersor balra történő léptetéseinek számát, ok tális számokban. Az 5-oktális számjegyű részprogram címnek a B(ADR) részregiszterből az A 22...36 részregiszterbe történő átléptetését a C1 számlálónak 5-re beállításával, és azután ennek egészen 0-ig való visszaszámláltatásával vezérlik. A C2 számláló számlálja a B regiszter balra lépéseinek számát, byte-okban. Úgy jelzik a következő szónak a bemeneti pufferből a B(ON) részregiszterbe való beolvasását, hogy a C2 12-es értéket kap, és azután ezt visszafelé számláltatják egészen a 0-ig. A C3 számláló azoknak a byte-oknak a számát számlálja, amelyeket az A regiszterbe léptetnek be, ahol a gépi utasítást szerkesztik össze. Az öt balra léptetést, amely ahhoz szükséges, hogy kialakítsák az első gépi utasítást az A regiszterben, a C3 számlálónak 7-es értékre állításával, és azután 12-ig való számláltatásával vezérlik.

11.3.2. Lehívási sorozat

A lehívási sorozat, amint a 11.10. ábrán látható, négy feladatot hajt végre. A B(ON) részregiszterben levő szót annyi oktális számjegypozícióval lépteti balra, amennyit a C1 számláló mutat, és így a legközelebbi áthelyezhető szó balra igazítva jelenik meg a B regiszterben. E balra léptetés az AB regisztersorban egyúttal átlépteti a B regiszter bal oldalán levő címet vagy adatot az A regiszterbe. Ezután átviszik a B(1—3), B(4—7) és B(8—9) részregiszterekben levő OP, FS és II mezők tartalmát az OP, FS, ill. II regiszterekbe. Minthogy e három mezőre a B(1—9) részregiszterben a továbbiakban nincs szükség, a B regisztert három oktális számjegypozícióval balra léptetik, és így az áthelyezhető szó FLD mezeje balra igazított lesz a B regiszterben.

A fenti feladatokban két balra léptetés van; egy a B regiszterben és egy az AB regisztersorban. Ezt a két különböző léptetést az LP regiszter vezérli, amely léptetés típusától függően 0-t, ill. 1-et tartalmazhat. Ilyen balra léptetésre van szükség a címmódosítási sorozatban is. A kényelem kedvéért létrehoznak egy *szétválasztási részsorozatnak* nevezett részsorozatot. Ezt „hívják be" akkor, amikor az UNPACK regiszter 1-es értéket kap és „visszatér" a hívó sorozathoz, ha a részsorozatban az UNPACK regisztert 0-ra állítják vissza. Miután az UNPACK regiszter 1-es értéket kapott, a sorozat állandóan megvizsgálja ennek értékét és várja, hogy a regiszter tartalma ismét 0 legyen. Amikor az UNPACK regiszter 1-es értéket kap, az LP regiszternek is 0-ás vagy 1-es értéket kell kapnia abból a célból, hogy kiválassza a két lehetséges balra léptetés közül az egyiket.

.3.
..étválasztási sorozat

A 11.11. ábrán látható szétválasztási sorozatnak négy feladata van. Az első feladat hajtja végre a balra léptetést; ezt az alábbi feltételes mikroutasítás írja le:

IF(LP=0) THEN (B←3 shl B) ELSE (AB←3 shl AB)

és dekrementálja a C1 számlálót egészen addig, amíg értéke el nem éri a 0-t. Amikor a C1 számláló 0 lesz, az UNPACK regiszter visszaállítódik 0-ra. A második feladat az, hogy egy szót olvasson ki az X2 regiszter által megjelölt címen levő bemeneti pufferből, és azt a B(ON) részregiszterben helyezze el; ezt a C2 számláló vezérli (kezdetben 12-es értéket kap). Amikor aC2 számláló eléri a 0-t, végrehajtódott a szó kiolvasása a bemeneti pufferterületről. A harmadik feladat egy, az A regiszterben összeszerkesztett gépi utasítás tárolása az X3 regiszter által meghatározott címen levő kimeneti pufferterületbe; ezt a C3 regiszter vezérli (kezdeti értéke 0). Amikor a C3 számláló eléri a 12-t, végrehajtódott az A regiszterben levő összeszerkesztett utasítás tárolása. Ez logikailag csak azután történhet, hogy a C1 nulla lett. A negyedik feladat a várakozás. Amint a 11.11. ábrán látható, van egy várakozási hurok, amelyben az UNPACK részsorozat állandóan vizsgálja az UNPACK regisztert, és mindaddig várakozik, amíg ez utóbbi tartalma 0 nem lesz.

Megjegyezzük, hogy az UNPACK részsorozat behívásához a UNPACK regiszternek 1-es értéket kell kapnia, az LP regiszternek 0-nak vagy l-nek kell lennie, és a C1 számlálót a biztos kezdeti értékre kell beállítani.

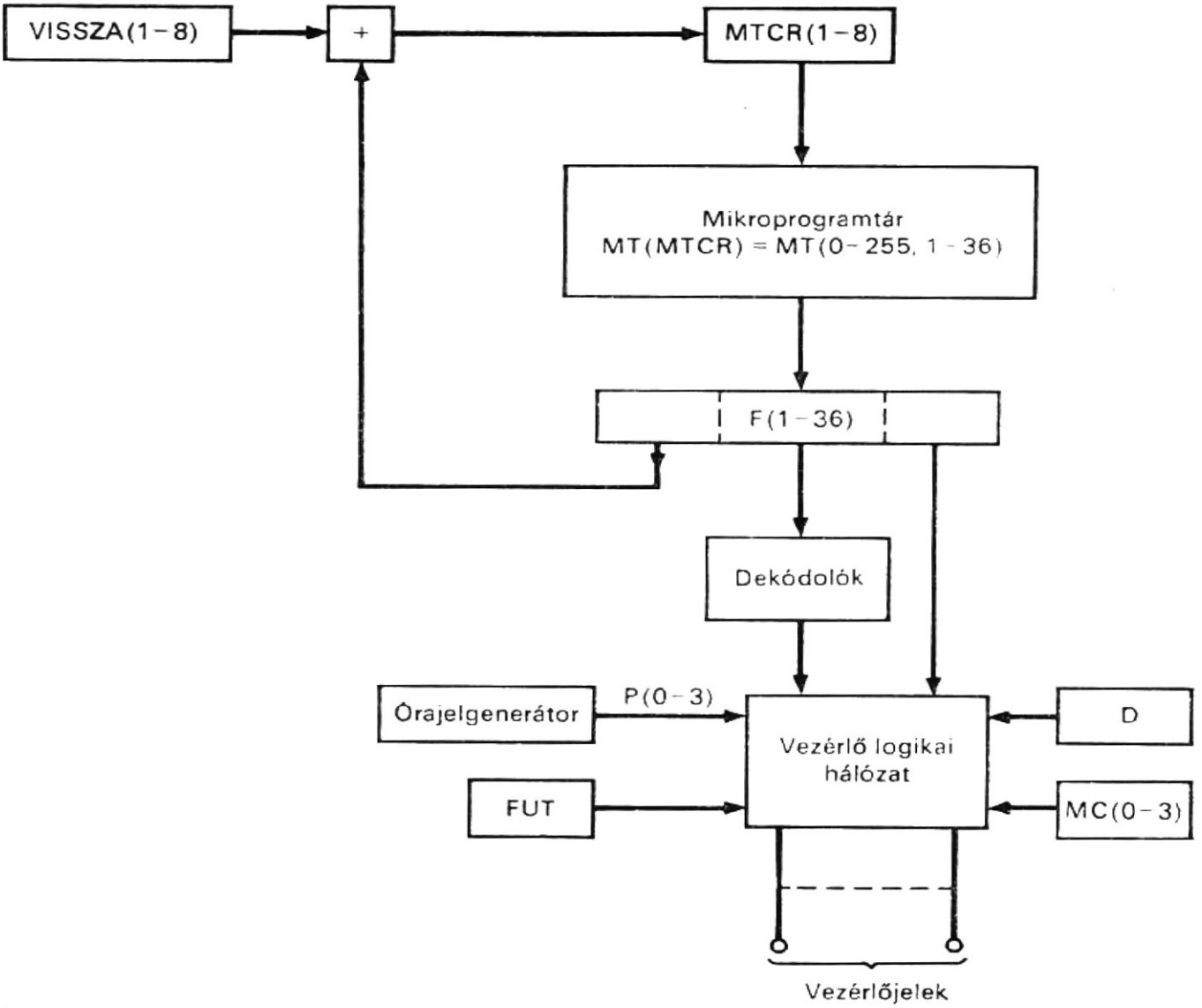
11.3.4.
Címmódosítási sorozat

A 11.12. ábrán levő címmódosítási sorozat, az OP és a II mezők által specifikált műveletet hajtja végre az FLD és az INC mezőkben levő operandusokon. Ha az OP mező tartalma 1, nincs címmódosítás. Ha az OP mező 2-t tartalmaz, a B(ADR) részregiszterben levő címet megnövelik az X7 indexregiszterben tárolt részprogram címmel. Az OP mező akár 1, akár 2, az FS mező tartalmát átviszik a C1 számlálóba.

Ha az OP mező értéke 3, a B(ADR) részregiszterben levő indexet megnövelik a közös szimbólumtáblázatnak az X4 regiszterben található helyével. A szót kiolvassák erről a tárhelyről, és az X8 regiszterben helyezik el. Ha az OP mező tartalma 4, a B(ADR) részregiszterben található indexet a definiálatlan szimbólumtáblázatnak az X5 regiszterben tárolt helyével növelik meg. Ezen a helyen egy másik cím van. Ezt a címet olvassák ki a tárból, tárolják az X8 regiszterben, és növelik meg a definiált szimbólumtáblázatnak az X6 regiszterben levő helyével. Azután ennek a tárhelynek a tartalmát olvassák ki a tárból és tárolják az X8 regiszterben.

Ha az OP mező tartalma 3 vagy 4, és az II mező nem 0, összeadást vagy kivonást is végezni kell. Először beléptetik a B(ADR) részregiszterbe az INC mező tartalmát, és azután ezt az X8 regiszter tartalmához hozzáadják (ha az II értéke 1), vagy abból kivonják (ha az II értéke 2), majd az eredményt a B(ADR) részregiszterben tárolják. A kivonás a kivonandó 2-es komplemensének hozzáadásával történik. Akár 3, akár 4 az OP mező tartalma, az FS mező tartalmát átviszik a C1 számlálóba.

Ezzel a címmódosítási sorozat befejeződik, és a folyamat visszatér a lehívási sorozathoz.



11.13. ábra
Mikroprogramozott vezérlés struktúrája

```
/MC(1)*P(3)*FUT/          $a mikroprogramtár-ciklus vége
/MC(2)*P(0)*FUT/          $a mikroprogramtár-ciklus kezdete
/MC(2)*P(1)*FUT/
/MC(2)*P(2)*FUT/
/MC(2)*P(3)*FUT/          $a mikroprogramtár-ciklus vége
/MC(3)*P(0)*FUT/          $a mikroprogramtár-ciklus kezdete
/MC(3)*P(1)*FUT/
/MC(3)*P(2)*FUT/
/MC(3)*P(3)*FUT/ D←0,$mindkét tárciklus vége
```

A fenti címkékben az egyes mikroprogramtár-ciklusokban a négy lépést a P(0—3) órajel négy fázisa vezérli, míg az egyes főtárciklusokban a négy mikroprogramtár-ciklus vezérlését az MC(0—3) gyűrűs számláló négy állapota látja el. A FUT regiszter feladata a vezérlőjelek aktiválása a főtárciklus 16 lépéséhez.

Minden egyes főtárciklus alatt kiolvasnak vagy beírnak egy utasítást a főtárból(ba). A vezérlés szervezése olyan, hogy a főtárcím átvitelének a CR regiszterbe, valamint a főtárból történő olvasás vagy a tárba írás időzítése a második lépés időpontjában történik (azaz /MC(0)*P(1)*FUT/). Olvasási művelet céljára a szó a TR tartalom-(puffer)-regiszterben áll rendelkezésre a hatodik lépésben (azaz, /MC(1)*P(1)*FUT/). Írási műveletkor a tárban elhelyezendő szót a TR tartalom-(puffer-)regiszterbe a tizenkettedik lépés előtt írják be (/MC(2)*P(3)*FUT/).

Ha néhány mikroművelet minden egyes mikroprogram-tárciklusban előfordul, a következő, négy címkéből álló sorozatot használják:

Comment, vezérlőjelek egy mikroprogramtár-ciklusban (11.4)

```
/P(0)*FUT*D'/ F←MT(MTCR)   $a mikroprogramtár-ciklus kezdete
/P(1)*FUT*D'/
/P(2)*FUT*D'/
/P(3)*FUT*D'/              $a mikroprogramtár-ciklus vége
```

A fentiekben a D regisztert használják a mikroprogramtár-ciklusban a négy lépés előrehaladásának vagy leállásának vezérlésére. Ha a D regiszter tartalma 0, a címkék aktívak, egyébként pedig eltűnnek.

Minden egyes mikroprogramtár-ciklusban egy mikroutasítás kerül kiolvasásra a mikroprogramtárból. Úgy időzítik, hogy a mikroprogramtár-címnek az MTCR regiszterbe való átvitele és a mikroprogramtár-kiolvasás indítása a megelőző mikroprogramtár-ciklus P(3) órajelfázisában megy végbe, míg a mikroutasítás az F tartalom-(puffer-) regiszterben a pillanatnyi mikroprogram-tárciklus P(0) első órajelfázisában fog rendelkezésre állni. Az F regiszterben levő mikroutasítás által aktivált mikroműveleteket a pillanatnyi mikroprogramtár-ciklus P(1—3) órajelfázisaiban hajthatják végre.

A D regiszter automatikusan 0-ra állítódik minden egyes főtárciklus végén (/MC(3)*P(3)*FUT/). Tehát ha várni kell a főtárciklus kezdetére, a D regiszternek 1-es értéket adnak, ezzel a vezérlőjelek generálását leállítják a mikroprogramtár-ciklus alatt, míg a főtárciklus vezérlőjelei folytatódnak. Ha egy mikroutasítást hívnak le a főtárciklus kezdetén, és ugyanakkor a D regiszter 1-es értéket kapott, akkor a mikroutasítás egy főtárciklusra az F regiszterben marad, amint azt a későbbiekben le fogjuk írni.

11.4.3. Mikroutasítás-formátum

A 11.1. táblázatban láthatjuk a mikroutasítás formátumát. Az egyes mikroutasítások hossza 36 bit; az F regiszter három mezőre osztható:

1. F(1—8) mező, amely a mikroprogramtár-címet tartalmazza.
2. F(9—23) mező, ez öt részmezőre osztódik, az egyes részmezőkhöz kapcsolódó dekódolókkal.
3. F(24—36) mezőben minden egyes bit egy mikroműveletet vagy mikroműveletek egy csoportját vezérli. Az F(9—36) mezőben 28 vezérlőbit van, ezek 43 végrehajtási utasítást vezérelnek.

Az F(1—8) mező kétutas elágazást szolgáltat mindegyik mikroutasításhoz. Az öt részmező az F(9—23) mezőben a következő: F(9—11), F(12—13), F(14—16), F(17—20), F(21—23). Mindegyik részmező olyan mikroműveleteket vezérel, amelyek előfordulása kölcsönösen kizárt. Az F(9—11) mező a három számlálóhoz tartozó mikroműveleteket vezérli. Az F(12—13) mező a számlálási mikroműveleteket vezérli. Az F(14—16) mező azokat a mikroműveleteket vezérli, amelyek a CR címregisztert működtetik. Az F(17—20) mező a TR tartalomregisztert és a B regisztert érintő mikroműveleteket vezérli. Az F(21—23) mező vezérli azokat a mikroműveleteket, amelyek a mikroprogramtár új címét írják be az MTCR regiszterbe.

A 13 vezérlőbit az F(24—36) mezőben a fennmaradó mikroműveleteket vezérli. Figyeljük meg, hogy az F(24) bit vezérli a B regisztert, és az AB regisztersorhoz tartozó léptető és vizsgáló mikroműveleteket a két, P(1) és P(3) órajelfázisban.

11.1. táblázat

Mikroutasítás
Vezérlőszó formátuma

Vezérlő-bitek	Dekódoló	Vezérlőjel	Mikroműveletek
F(1—8)			Mikroprogramtár-címmező
F(9—11)	DC(0—7)		
	DC(1)	D′ * P(1)	C1 ← FS,
	DC(2)	D′ * P(1)	C1 ← 3,
	DC(3)	D′ * P(1)	C1 ← 5,
	DC(4)	D′ * P(1)	C2 ← 12,
	DC(5)	D′ * P(1)	C3 ← 0,
	DC(6)	D′ * P(1)	C3 ← 7,
F(12—13)	DX(0—3)		
	DX(1)	D′ * P(2)	X1 ← countup X1,
	DX(2)	D′ * P(2)	X2 ← countup X2,
	DX(3)	D′ * P(2)	X3 ← countup X3,
F(14—16)	DAR(0—7)		
	DAR(1)	MC(0) * P(1)	CR ← X2,
	DAR(2)	MC(0) * P(1)	CR ← X3,
	DAR(3)	MC(0) * P(1)	CR ← X8 add X4,
	DAR(4)	MC(0) * P(1)	CR ← X8 add X5,
	DAR(5)	MC(0) * P(1)	CR ← X8 add X6,
F(17—20)	DBS(0—15)		
	DBS(1)	D′ * P(2)	TR ← A,
	DBS(2)	MC(2) * P(1)	B(ON) ← TR,
	DBS(3)	MC(2) * P(1)	X8 ← TR(AD),
	DBS(4)	D′ * P(1)	X1 ← B(ADR),
	DBS(5)	D′ * P(1)	B(ADR) ← X7,
	DBS(6)	D′ * P(1)	X8 ← B(ADR),
	DBS(7)	D′ * P(1)	B(ADR) ← X8 add X1,
	DBS(8)	D′ * P(1)	B(ADR) ← X8 add X7,
F(21—23)	DT(0—7)		
	DT(1)	D′ * P(3)	MTCR ← countup MTCR,
	DT(2)	D′ * P(3)	MTCR ← F(CMK),
	DT(3)	D′ * P(3)	MTCR ← F(1—5)—OP,
	DT(4)	D′ * P(3)	MTCR ← VISSZA,
	DT(5)	D′ * P(3)	IF (II=0) THEN (MTCR ← F(CMK)) ELSE (MTCR ← countup MTCR),
	DT(6)	D′ * P(3)	IF (II=1) THEN (MTCR ← F(CMK)) ELSE (MTCR ← countup VCR),
	DT(7)	MC(2) * P(3)	MTCR ← countup MTCR,
		MC(3) * P(3)	IF (C1≠0) THEN (MTCR ← F(CMK)) IF ((C1≠0) * (C3≠12)) THEN (MTCR ← VISSZA),

11.1. táblázat folytatása

Vezérlő-bitek	Dekódoló	Vezérlőjel	Mikroműveletek

F(24)		D′ * P(1)	IF (LP=0) THEN (B←3 shl B)
			ELSE (AB←3 shl AB),
			IF (LP=1) THEN (C3→countup C3),
			C1←countdn C1,
			C2←countdn C2,

		D′ * P(3)	IF (C2=0) THEN (MTCR←countup MTCR,
			IF (MC(0)+MC(1)+MC(2)=1)
			THEN (D←1)),
			IF ((C1=0) * (C2≠0) * (C3=12))
			THEN (MTCR←F(CMK),
			IF (MC(0)+MC(1)+MC(2)=1)
			THEN (D←1)),
			IF ((C1=0) * (C2≠0) * (C3≠12))
			THEN (MTCR←VISSZA),

F(25)		D′ * P(1)	OP←B(1—3), FS←B(4—7), II←B(8—9).
F(26)		D′ * P(2)	VISSZA←MTCR,
		D′ * P(3)	VISSZA←countup VISSZA,
F(27)		D′ * P(1)	A←0,
F(28)		D′ * P(1)	X1←0
F(29)		D′ * P(1)	X1←X1′,
F(30)			(nem használják)
F(31)		MC(0) * P(1)	OLVAS←1
F(32)		D′ * P(2)	LP←0,
F(33)		D′ * P(3)	IF (MC(0)+MC(1)+MC(2)=1) THEN (D←1),
F(34)		MC(0) * P(1)	IR←1,
F(35)		D′ * P(2)	LP←1,
F(36)		D′ * P(3)	FUT←0,

11.5. Mikroprogramozás

Ebben a fejezetben bemutatjuk a betöltő mikroprogramozását. A mikroprogramot először CDL utasításokkal írjuk meg, ebből állítjuk elő a mikroprogram kódolt alakját. A CDL utasításokban leírt mikroprogram a 11.9...11.12. folyamatábrákból és a 11.1. táblázatban levő vezérlőszó-formátumból határozhatók meg. Amint látni fogjuk, a mikroprogram 24 mikroutasításból áll: három a szétválasztási sorozathoz tartozik, kettő az iniciálási sorozathoz, kettő a lehívási sorozathoz és 16 a címmódosítási sorozathoz.

11.5.1.
Szétválasztási sorozat

Az alábbiakban leírjuk a sorozat három mikroutasítását.

```
Comment, szétválasztási sorozat                                    (11.5)
Comment, léptetés és teszt mikroutasítás, az MT 63. címén elhelyezve
/D'*FUT*P(0)/                F←MT(MTCR),
/D'*FUT*P(1)*F(24)/          IF (LP=0) THEN (B←3 sh1 B)
                              ELSE (AB←3 shl AB),
                             IF (LP=1) THEN (C3←countup C3),
                             C1←countdn C1,
                             C2←countdn C2,                        (11.6)
/D'*FUT*P(3)*F(24)/          IF (C2=0)
                              THEN (MTCR←countup MTCR,
                              IF (MC(0)+MC(1)+MC(2)=1)
                              THEN (D←1)),
                             IF ((C1=0)*(C2≠0)*(C3=12))
                              THEN (MTCR←F(CMK),
                              IF (MC(0)+MC(1)+MV(2)=1)
                              THEN (D←1)), $F(CMK)=65
                             IF ((C1=0)*(C2≠0)*(C3≠12))
                              THEN (MTCR←VISSZA)

Comment, főtárszó betöltését végző mikroutasítás, az MT 64. címén elhelyezve (11.7)
/D'*FUT*P(0)/                F←MT(MTCR),
/FUT*MC(0)*P(1)*DAR(1)/      CR←X2,
/FUT*MC(0)*P(1)*F(31)/       OLVAS←1,
/D'*FUT*P(1)*DC(4)/          C2←12,
/D'*FUT*P(2)*DX(2)/          X2←countup X2,
/D'*FUT*P(3)*F(33)/          IF (MC(0)+MC(1)+MC(2)=1)
                              THEN (D←1),

Comment, ez a mikroutasítás az F-ben marad, a főtárciklus végéig
/FUT*MC(1)*P(1)/             TR←M(CR),
/FUT*MC(2)*P(1)DBS(2)/       B(ON)←TR,
/FUT*MC(2)*P(3)*DT(7)/       MTCR←countup MTCR,
/FUT*MC(3)*P(3)*DT(7)/       IF (C1≠0)
                              THEN (MTCR←F(CMK)), $F(CMK)=63
                             IF ((CL=0)*(C3≠12))
                              THEN (MTCR←VISSZA),
/FUT*MC(3)*P(3)/             D←0,

Comment, egy főtárszó tárolása (store) mikroutasítás, az MT 65. címén elhelyezve
                                                                   (11.8)
/D'*FUT*P(0)/                F←MT(MTCR),
/FUT*MC(0)*P(1)*DAR(2)/      CR←X3,
/FUT*MC(0)*P(1)*F(34)/       IR←1,
/D'*FUT*P(1)*DC(5)/          C3←0,
/D'*FUT*P(2)*DX(3)/          X3←countup X3,
/D'*FUT*P(2)*DBS(1)/         TR←A,
/D'*FUT*P(3)*DT(4)/          MTCR←VISSZA,
/FUT*MC(3)*P(1)/             M(CR)←TR,
```

A fenti leírásban a 11.11. ábra UNPACK regiszterét a VISSZA regiszterrel helyettesítettük. Amikor behívják a szétválasztási sorozatot, a mikroprogramtár következő címét a VISSZA regiszter tárolja, míg a behívó sorozattól az átvitelt a szétválasztási sorozathoz az MTCR←F(CMK) mikroművelet végzi el. Az F(CMK) részregiszter tartalmazza a szétválasztási sorozat címét a mikroprogramtárban. Amikor a szétválasztási sorozat befejeződik, a behívó sorozathoz a visszatérést az MTCR←VISSZA mikroművelet hajtja végre.

Amint a fentiekben látható, az első mikroutasítás a mikroprogramtár 63. címén helyezkedik el. Amint a 11.11. folyamatábrából is kitűnik, az AB regisztersor B regiszterének léptetése mellett ez a mikroutasítás hajtja végre az alábbi négyutas elágazást.

1. Ha a (C1≠0)∗(C2≠0) feltétel igaz, akkor meg kell ismételni a léptetés és teszt mikroutasítást.
2. Ha a (C2=0) feltétel igaz, akkor egy, a bemeneti pufferterületen levő szót kiolvasnak a főtárból és a B(ON) részregiszterbe helyezik el.
3. Ha a (C1=0)∗(C2≠0)∗C3=12) feltétel igaz, akkor az A regiszter tartalmát beírják a főtár tartalomregiszterébe.
4. Ha a (C1=0)∗(C2≠0)∗(C3≠12) feltétel igaz, akkor a szétválasztási részsorozat befejeződött és a vezérlés visszatér a behívó sorozathoz.

Ha a 2. vagy 3. elágazás kerül végrehajtásra, a D regiszter 1 értéket kap, mivel a végrehajtandó mikroutasítások egy főtárciklust igényelnek.

A második mikroutasítás a mikroprogramtár 64. címén helyezkedik el. Akkor hívják le és hajtják végre, ha a léptetés és teszt mikroutasítás (C2=0) feltétele igaz. Ez a mikroutasítás egy szót olvas ki a főtárban levő bemeneti pufferterületről, és azt a B(ON) részregiszterbe tárolja. A mikroutasítás egy főtárciklust igényel. Ennek megvalósításához a D regiszter 1 értéket kap. Így a mikroutasítás az F regiszterben marad a teljes főtárciklusra. E mikroprogramtár-ciklus folyamán kezdeményezik a beírást a kimeneti pufferterületre és három mikroprogramtár-ciklussal később fejeződik be. Ezúttal a mikroutasítás ismét visszaadja a vezérlést a hívósorozatnak.

11.5.2. Iniciálási sorozat

Ez a műveletsorozat iniciálja a betöltőt. A sorozatot képező három mikroutasítást az alábbiakban mutatjuk be.

```
Comment, iniciálási sorozat                                        (11.9)

Comment, a bemeneti puffer olvasását kezdeményező mikroutasítás,
az MT 66. címén                                                    (11.10)

/D'∗FUT∗P(0)/                F←MT(MTCR),
/FUT∗MC(0)∗P(1)∗DAR(1)/      CR←X2,
/FUT∗MC(0)∗P(1)∗F(31)/       OLVAS←1,
/D'∗FUT∗P(1)∗DBS(5)/         B(ADR)←X7,
/D'∗FUT∗P(1)∗DC(6)/          C3←7
/D'∗FUT∗P(1)∗F(27)/          A←0,
/D'∗FUT∗P(3)∗DT(1)/          MTCR←countup MTCR,

Comment, főtárból történő olvasás, mikroutasítás, az MT 67. címén (11.11)

/D'∗FUT∗P(0)/                F←MT(MTCR),
```

```
/FUT * MC(1) * P(0)/          TR←M(MTCR),
/D' * FUT * P(1) * DC(4)/     C2←12,
/D' * FUT * P(2) * DX(2)/     X2←countup X2
/D' * FUT * P(3) * DT(1)/     MTCR←countup MTCR,
```

Comment, a B regisztert betöltő mikroutasítás, az MT 68. címén elhelyezve

```
/D' * FUT * P(0)/             F←MT(MTCR),
/FUT * MC(2) * P(1) * DBS(2)/ B(ON)←TR,
/D' * FUT * P(1) * DC(3)/     C1←5,
/D' * FUT * P(3) * DT(1)/     MTCR←countup MTCR,
```

A mikroprogramtár 66., 67. és 68. címén elhelyezett három mikroutasítás szekvenciálisan hajtja végre a műveleteket a főtárciklus első három mikroprogramtár ciklusában. Ezek határozzák meg az A és B regiszter tartalmát és a már meghatározott értékkel töltik fel a számlálókat a lehívási sorozat előkészítéséhez. A három mikroutasítás inkrementálja az MTCR mikroprogramtár-címregisztert, a lehívási sorozat megkezdéséhez.

11.5.3. Lehívási sorozat

Ez a műveletsorozat hívja le a következő áthelyezhető szót. A lehívási műveletsorozat két mikroutasítása a következő:

Comment, lehívási műveletsorozat (11.12)

Comment, szétválasztási műveletsorozat indítása mikroutasítás, az MT 69. címén

```
/D' * FUT * P(0)/             F←MT(MTCR),                    (11.13)
/D' * FUT * P(2) * F(35)/     LP←1,
/D' * FUT * P(2) * F(26)/     VISSZA←MTCR,
/D' * FUT * P(3) * F(26)/     VISSZA←countup VISSZA,
/D' * FUT * P(3) * DT(2)/     MTCR←F(CMK), $F(CMK)=63
```

Comment, dekódolás és szétválasztási műveletsorozat indítása mikroutasítás, az MT 70. címén (11.14)

```
/D' * FUT * P(0)/             F←MT(MTCR),
/D' * FUT * P(1) * F(25)/     OP←B(1—3), FS←B(4—7),
                              II←B(8—9),
/D' * FUT * P(1) * DC(2)/     C1←3,
/D' * FUT * P(2) * F(32)/     LP←0,
/D' * FUT * P(2) * F(26)/     VISSZA←MTCR,
/D' * FUT * P(3) * F(26)/     VISSZA←countup VISSZA,
/D' * FUT * P(3) * DT(2)/     MTCR←F(CMK), $F(CMK)=63
```

Amint a fentiekben látható, az első mikroutasítás a mikroprogramtár 69. címén helyezkedik el. Ez a mikroutasítás vezérli az átmenetet a lehívási műveletsorozatból a szétválasztási műveletsorozathoz, vezérlést adva az AB regisztersor léptetésének elvégzésére, ami a következő mikroutasítás címének a VISSZA regiszterbe töltésével, az LP regiszternek 1-es érték adásával és a vezérlési címregiszterbe a szétválasztási sorozat címének betöltésével megy végbe.

A második mikroutasítást a szétválasztási műveletsorozatból való visszatéréskor hajtják végre. Ez a mikroutasítás tölti be az OP, FS és II regisztereket, majd behívja a szétválasztási műveletsorozatot, miközben az LP regiszternek nulla értéket ad, hogy balra igazítsa a B regiszterben található áthelyezhető szó címrészét vagy adatrészét. Betölti a VISSZA regiszterbe a következő mikroutasítás címét, amely a címmódosítási műveletsorozat első mikroutasítása.

11.5.4. Címmódosítási sorozat

A címmódosítási utasítássorozatot a 11.13. folyamatábra szerinti alakban írjuk le. Az utasítássorozat CDL leírása:

Comment, címmódosítási utasítássorozat (11.15)

Comment, elágazás az MT 71. címén elhelyezkedő OP mikroutasításra (11.16)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/D'*FUT*P(1)*DBS(6)/         X8←B(ADR),
/D'*FUT*P(3)*DT(3)/          MTCR←F(1—5)—OP, $F(CMK)=72
/D'*FUT*P(3)*F(33)/          IF (MC(0)+MC(1)+MC(2)=1)
                              THEN (D←1),
```

Comment, hiba—stop mikroutasítás, amely az MT 72. címén helyezkedik el (11.17)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/D'*FUT*P(3)*F(36)/          FUT←0,
```

Comment, adat mikroutasítás, amely az MT 73. címén helyezkedik el (11.18)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/D'*FUT*P(1)*DC(1)/          C1←FS,
/D'*FUT*P(3)*DT(2))          MTCR←F(CMK), $F(CMK)=69
```

Comment, relatívcím mikroutasítás, amely a VT 74. címén helyezkedik el (11.19)

```
/D'*FUT*P(0)                 F←MT(MTCR),
/D'*FUT*P(1)*DBS(8)/         B(ADR)←X8 add X7,
/D'*FUT*P(1)*DC(1)/          C1←FS,
/D'*FUT*P(3)*DT(2)/          MTCR←F(CMK), $F(CMK)=69
```

Comment, közösterületcím mikroutasítás, amely az MT 75. címén helyezkedik el (11.20)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/FUT*MC(0)*P(1)*DAR(3)/      CR←X8 add X4,
/FUT*MC(0)*P(1)*F(31)/       OLVAS←1,
/D'*FUT*P(3)*DT(2)/          MTCR←F(CMK), $F(CMK)=81
```

Comment, külsőcím mikroutasítás, amely az MT 76. címén helyezkedik el (11.21)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/FUT*MC(0)*P(1)*DAR(4)/      CR←X8 add X5,
/FUT*MC(0)*P(1)*F(31)/       OLVAS←1,
/D'*FUT*P(3)*DT(2)/          MTCR←F(CMK), $F(CMK)=80
/D'*FUT*P(3)*F(33)/          IF (MC(0)+MC(1)+MC(2)=1)
                              THEN (D←1),
```

```
/FUT * MC(1) * P(1)/                TR←M(CR),
/FUT * MC(2) * P(1) * DBS(3)/       X8←TR(CM),
/FUT * MC(3) * P(3)/                D←0,
```

Comment, stop mikroutasítás, amely az MT 77. címén helyezkedik el (11.22)

```
/D' * FUT * P(0)/                   F←MT(MTCR),
/D' * FUT * P(3) * F(36)/           FUT←0,
```

Comment, hiba-stop mikroutasítás, amely az MT 78. címén helyezkedik el (11.23)

```
/D' * FUT * P(0)/                   F←M(MTCR),
/D' * FUT * P(3) * F(36)/           FUT←0,
```

Comment, hiba-stop mikroutasítás, amely az MT 79. címén helyezkedik el (11.24)

```
/D' * FUT * P(0)/                   F←MT(MTCR),
/D' * FUT * P(3) * F(36)/           FUT←0,
```

Comment, főtártáblázat olvasási mikroutasítás, amely az MT 80. címén helyezkedik el (11.25)

```
/D' * FUT * P(0)/                   F←MT(MTCR),
/FUT * MC(0) * P(1) * DAR(5)/       CR←X8 add X6,
/FUT * MC(0) * P(1) * F(31)/        OLVAS←1,
/D' * FUT * P(3) * DT(1)/           MTCR←countup MTCR,
```

Comment, elágazás ha nincs inkrementálás mikroutasítás, amely az MT 81. címén helyezkedik el (11.26)

```
/D' * FUT * P(0)/                   F←MT(MTCR),
/FUT * MC(1) * P(1)/                TR←M(CR),
/D' * FUT * P(3) * DT(5)/           IF (II=0)
                                      THEN (MTCR←F(CMK))
                                      ELSE (MTCR←countup MTCR),
                                      $F(CMK)=86
```

Comment, szétválasztási sorozat indítása mikroutasítás, amely az MT 82. címén helyezkedik el (11.27)

```
/D' * FUT * P(0)/                   F←MT(MTCR),
/FUT * MC(2) * P(1) * DBS(3)/       X8←TR(CM),
/D' * FUT * P(1) * DC(1)/           C1←FS,
/D' * FUT * P(2) * F(32)/           LP←0,
/D' * FUT * P(2) * F(26)/           VISSZA←MTCR,
/D' * FUT * P(3) * F(26)/           VISSZA←countup VISSZA,
/D' * FUT * P(3) * DT(2)/           VCR←F(CMK), $F(CMK)=63
```

Comment, inkrementálás—dekrementálás elágazási mikroutasítás, amely az MT 83. címén helyezkedik el (11.28)

```
/D' * FUT * P(0)/                   F←MT(MTCR),
/D' * FUT * P(1) * DBS(4)/          X1←B(ADR)
/D' * FUT * P(3) * DT(6)/           IF (II=1)
                                      THEN (MTCR←F(CMK))
                                      ELSE (MTCR←countup MTCR),
                                      $F(CMK)=85
```

Comment, 2-es komplemens mikroutasítás, amely az MT 84. címén helyezkedik el (11.29)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/D'*FUT*P(1)*F(29)/          X1←X1',
/D'*FUT*P(2)*DX(1)/          X1←Countup X1,
/D'*FUT*P(3)*DT(1)/          MTCR←countup MTCR,
```

Comment, abszolútcím-módosítás mikroutasítás, amely az MT 85. címén helyezkedik el (11.30)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/D'*FUT*P(1)*DBS(7)/         B(ADR)←X8 add X1,
/D'*FUT*P(1)*DC(1)/          C1←FS,
/D'*FUT*P(3)*DT(2)/          MTCR←F(CMK), $F(CMK)=69
```

Comment, nincs inkrementálás mikroutasítás, amely az MT 86. címén helyezkedik el (11.31)

```
/D'*FUT*P(0)/                F←MT(MTCR),
/FUT*MC(2)*P(1)*DBS(3)/      X8←TR(CM),
/D'*FUT*P(1)*F(28)/          X1←0,
/D'*FUT*P(3)*DT(2)/          MTCR←F(CMK), $F(CMK)=85
```

A (11.16) mikroutasítás a mikroprogramtár 71. címén helyezkedik el. Ez a mikroutasítás nyolcutas elágazást hajt végre, az OP regiszter tartalmától függően. Ez úgy megy végbe, hogy az F(CMK) részregiszter első öt bitjét összefűzik (konkatenálják) az OP regiszterrel, egy nyolcbites mikroprogramtár-cím létrehozására. A nyolc lehetséges cím közül öt legális; ezeket az alábbiakban részletesen megtárgyaljuk. A másik három, amely 0, 6 és 7 OP-értékeknek felelnek meg, hiba-stopot okoz. Az F(CMK) részregiszter tartalmazza a mikroprogramtár 72. címét. Ez azt jelenti, hogy a nyolc lehetséges cím 72...79., ahol a 72., 78. és 79. az illegális címeknek felelnek meg. Megjegyezzük, hogy az X8 regiszterbe töltik be ezt a mikroutasítást.

Ha az OP értéke 1, a 73. mikroprogramtár-címen levő (11.18) mikroutasítás kerül végrehajtásra. Ez azt jelenti, hogy az áthelyezhető szó adatot tartalmaz, tehát a mikroutasítás egyszerűen betölti a C1 számlálót és elágazik a lehívási sorozathoz.

Ha az OP értéke 2, relatív címet jelöl: a 74. mikroprogramtár-címen levő (11.19) mikroutasítás kerül végrehajtásra. Ez a mikroutasítás hozzáadja az X8-ban levő címet az X7-ben található részprogramcímhez és elágazik a lehívási sorozathoz.

Ha az OP értéke 3, közös területű címet jelöl; a mikroprogramtár 75. címén levő (11.20) mikroutasítás kerül végrehajtásra. Ez a mikroutasítás a főtárban levő *közös szimbólumtáblázatnak* az olvasását kezdeményezi, és elágazik a 81. helyen levő mikroutasításhoz, annak megvizsgálására, hogy létezik-e inkrementáló mező az áthelyezhető szóban. Megjegyezzük, hogy a (11.16) mikroutasítás állítja be a D regisztert abból a célból, hogy biztosítsa ennek az utasításnak végrehajtását a főtárciklus kezdetekor.

Ha az OP értéke 4, ez külső címet jelöl; a mikroprogramtár 76. címén levő (11.21) mikroutasítás kerül végrehajtásra. Ez a mikroutasítás a főtárban levő *definiálatlan szimbólumtáblázatból* kiolvasást hajt végre, és ezután elágazik a mikroprogramtár 80. címén levő mikroutasításhoz, amely a főtárban található definiált szimbólumtáblázatból való kiolvasást kezdeményez.

Ha az OP értéke 5, ez a fordítási folyamat végét jelzi; a mikroprogramtár 77. címén levő (11.22) mikroutasítást hajtják végre. Ez a mikroutasítás nulla értéket ad a FUT regiszternek, valamennyi jel generálásának leállítására.

A mikroprogramtár 81...86. helyén levő, fennmaradó mikroutasításokat akkor hajtják végre, ha az OP 3-mal vagy 4-gyel egyenlő. Ezek a mikroutasítások végzik az inkrementálásra vagy dekrementálására való vizsgálatot, valamint a címmódosítási

műveleteket, amelyeket a 11.12. folyamatábra ír le. Ez a mikroprogramtár 82. címén, a (11.27) mikroutasításban a szétválasztási sorozat indítását jelenti. Megjegyezzük azt is, hogy a főtár kiolvasása, amit a mikroprogramtár 75. címén (OP=3) vagy 80. címén (OP=4) elhelyezkedő mikroutasítás kezdeményez, párhuzamosan folytatódik ezeknek a mikroutasításoknak a végrehajtásával.

11.5.5. Mikroprogram

A mikroprogram a 11.2. táblázatban látható, ahol a mikroutasítások a tetszőlegesen választott 63...86. helyen találhatók. A mikroprogram egyes mikroutasításait az előző CDL leírásból kaptuk.

Például azt a (11.7) mikroutasítást, amely egy főtárszó betöltésére szolgál és amelyet előzetesen a szétválasztási sorozatban írtunk le, a 64. helyhez rendeltük hozzá. Az F(CMK) mező 63-at kell, hogy tartalmazzon, a mikroutasításban levő MTCR← ←F(CMK) elágazási mikroművelet miatt. Minden egyes végrehajtási utasításban van egy mikroművelet és egy megfelelő dekódolókimenet vagy vezérlőbit a címkében, a 11.1. táblázatban látható mikroutasítás-formátum szerint. Nézzük meg e példa második végrehajtási utasítását. Ez az utasítás a CR—X2 mikroműveletet tartalmazza.

11.2. táblázat

A betöltő mikroprogramja

Vezérlő-tárcím	F(CMK)=F(1—8)	DC (0—7)	DX (0—3)	DAR (0—7)	DBS (0—15)	DT (0—7)	F(24—36)
63	0 1 0 0 0 0 0 1	0 0 0	0 0	0 0 0	0 0 0 0	0 0 0	1 0 0 0 0 0 0 0 0 0 0 0 0
64	0 0 1 1 1 1 1 1	1 0 0	1 0	0 0 1	0 0 1 0	1 1 1	0 0 0 0 0 0 0 1 0 1 0 0 0
65	0 0 0 0 0 0 0 0	1 0 1	1 1	0 1 0	0 0 0 1	1 0 0	0 0 0 0 0 0 0 0 0 0 1 0 0
66	0 0 0 0 0 0 0 0	1 1 0	0 0	0 0 1	0 1 0 1	0 0 1	0 0 0 1 0 0 0 1 0 0 0 0 0
67	0 0 0 0 0 0 0 0	1 0 0	1 0	0 0 0	0 0 0 0	0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0
68	0 0 0 0 0 0 0 0	0 1 1	0 0	0 0 0	0 0 1 0	0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0
69	0 0 1 1 1 1 1 1	0 0 0	0 0	0 0 0	0 0 0 0	0 1 0	0 0 1 0 0 0 0 0 0 0 0 1 0
70	0 0 1 1 1 1 1 1	0 1 0	0 0	0 0 0	0 0 0 0	0 1 0	0 1 1 0 0 0 0 0 1 0 0 0 0
71	0 1 0 0 1 0 0 0	0 0 0	0 0	0 0 0	0 1 1 0	0 1 1	0 0 0 0 0 0 0 0 0 1 0 0 0
72	0 0 0 0 0 0 0 0	0 0 0	0 0	0 0 0	0 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 1
73	0 1 0 0 0 1 0 1	0 0 1	0 0	0 0 0	0 0 0 0	0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0
74	0 1 0 0 0 1 0 1	0 0 1	0 0	0 0 0	1 0 0 0	0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0
75	0 1 0 1 0 0 0 1	0 0 0	0 0	0 1 1	0 0 0 0	0 1 0	0 0 0 0 0 0 0 1 0 0 0 0 0
76	0 1 0 1 0 0 0 0	0 0 0	0 0	1 0 0	0 0 1 1	0 1 0	0 0 0 0 0 0 0 1 0 1 0 0 0
77	0 0 0 0 0 0 0 0	0 0 0	0 0	0 0 0	0 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 1
78	0 0 0 0 0 0 0 0	0 0 0	0 0	0 0 0	0 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 1
79	0 0 0 0 0 0 0 0	0 0 0	0 0	0 0 0	0 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 1
80	0 0 0 0 0 0 0 0	0 0 0	0 0	1 0 1	0 0 0 0	0 0 1	0 0 0 0 0 0 0 1 0 0 0 0 0
81	0 1 0 1 0 1 1 0	0 0 0	0 0	0 0 0	0 0 0 0	1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0
82	0 0 1 1 1 1 1 1	0 0 1	0 0	0 0 0	0 0 1 1	0 1 0	0 0 1 0 0 0 0 0 1 0 0 0 0
83	0 1 0 1 0 1 0 1	0 0 0	0 0	0 0 0	0 1 0 0	1 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0
84	0 0 0 0 0 0 0 0	0 0 0	0 1	0 0 0	0 0 0 0	0 0 1	0 0 0 0 0 1 0 0 0 0 0 0 0
85	0 1 0 0 0 1 0 1	0 0 1	0 0	0 0 0	0 1 1 1	0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0
86	0 1 0 1 0 1 0 1	0 0 0	0 0	0 0 0	0 0 1 1	0 1 0	0 0 0 0 1 0 0 0 0 0 0 0 0

A mikroutasítás-formátumnak megfelelően ezt a mikroműveletet az F(14—16) mezőben levő DAR(1) dekódoló végérték vezérli; ezért ez a mező — amint a 11.2. táblázat 64. helyén látható — a 001 értéket tartalmazza. Hasonlóképpen, egy-egy nemnulla részmező vezérli a mikroművelet vagy mikroműveletek egy csoportjának végrehajtását. Vizsgáljuk meg a tárolvasást kezdeményező OLVAS←1 mikroműveletet is. A mikroutasítás-formátum szerint ezt a mikroműveletet az F(31) vezérlőbit vezérli. A 11.2. táblázat 64. helyén levő mikroutasításszó azt mutatja, hogy ez a bit 1. Hasonlóképpen az F(24—36) mezőben egy-egy vezérlőbit vezérli egy mikroművelet vagy mikroművelet-csoport végrehajtását. Ebben az értelemben készült a 11.2. táblázatban látható mikroprogram.

E mikroprogram 71. helyén a mikroutasítás elágazik a 72...79. helyek egyikére. A 72...79. helyek nyolc mikroutasítást tartalmaznak (csak ötöt használnak belőlük). Ezek mindegyike az áthelyezhető szó OP-kódja által igényelt mikroutasítást hajt végre. Ezt a táblázatot az F(CMK) mezőben levő cím határozza meg (l. a 11.2. táblázat 71. helyén a 01001000 értéket).

Mód van arra, hogy megbecsüljük a végrehajtható kód (vagyis utasítások és az adatszavak) fordításának sebességét. Az egyik szélső érték szerint hét főtárciklust igényel egy adatszó a lefordításához. A másik szélső érték szerint 24 főtárciklusra van szüksége egy kétcímes utasítás lefordításához, ahol a címek külső címek, mivel négy áthelyezhető szó kell egy ilyen utasításhoz. Legyen a főtár ciklusideje 1 μs. Ekkor a mikroprogramozott betöltő 41 700...143 000 utasítást vagy adatszót tud előállítani egy másodperc alatt.

Feladatok

11.1. Felhasználva a 8. fejezetben a mikroprogram optimalizálására bemutatott módszereket, határozzuk meg a mikroutasítás további lehetséges formátumait! Cél, hogy a mikroutasítás hossza és a mikroprogram hossza egyidejűleg minél kisebb legyen.

11.2. Felhasználva a 11.1. feladatban meghatározott mikroutasítást, készítsünk CDL nyelvű mikroprogramot az áthelyezhető kód fordításához!

11.3. Rajzoljunk folyamatábrát a 11.2. feladatban leírt mikroprogramhoz!

Irodalom

[1] FLORES, I.: *Computer Software.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1965.

[2] HASSETT, A.: *Computer Programming and Computer Systems.* New York, Academic Press, Inc., New York, 1967.

[3] MAURER, W. D.: *Programming: An Introduction to Computer Languages and Techniques.* San Francisco, Holden-Day, Inc., 1968.

[4] BARRON, D. W.: *Assemblers and Loaders.* New York, American Elsevier Inc., 1969.

[5] ROSIN, R. F.: *Supervisory and Monitor Systems.* Computing Surveys of the ACM, March, 1969, pp. 37...54.

[6] CHU, Y.—PARDO, O. R.—YEH, J.: *A Methodology for Unified Hardware-software Design.* Technical Report 70-107, Computer Science Center, University of Maryland, January, 1970.

[7] CHU, Y.: *Introduction to Computer Organization.* Englewood Cliffs, N. J., Prentice-Hall, Inc., 1970.

A könyvben gyakrabban használt angol kifejezések magyar megfelelője*

AC (**AC**umulator)	AC (**A**kkumulátor)
ADD (**ADD**ition)	összeadás
ADDR (**ADD**ress **R**egister)	CR (**C**ím**R**egiszter)
ALS (**A**ccumulator **L**eft **S**hift)	akkumulátor balra lép
AR (**A**ddress **R**egiszter)	CR (**C**ím**R**egiszter)
ARS (**A**ccumulátor **R**ight **S**hift)	akkumulátor jobbra lép
BUS (**BUS**)	S (**S**in)
CAC (**C**hannel **A**dress **C**ounter)	CCS (**C**satorna **C**ím **S**zámláló)
CAW (**C**hannel **A**dress **W**ord)	CCS (**C**satorna **C**ím **S**zó)
CCW (**C**hannel **C**ommand **W**ord)	CPSZ (**C**satorna **P**arancs **S**zó)
CH (**CH**annel)	CS (**CS**atorna)
CIL (**CI**rcular **L**eft Shift)	ciklikus léptetés balra
CLA (**CL**ear and **A**dd)	törlés és összeadás
CLS (**C**lear and **S**ubtract)	törlés és kivonás
CM (**C**ontrol **M**emory)	MT (**M**ikroprogram **T**ár)
CPU (**C**entral **P**rocessing **U**nit)	KFE (**K**özponti **F**eldolgozó **E**gység)
CST (**C**ommon **S**ymbol **T**able)	közös szimbólumtáblázat
CSW (**C**hannel **S**tatus **W**ord)	CSSZ (**C**satorna **S**tatus **SZ**ó)
CU (**C**ontrol **U**nit)	VE (**V**ezérlő **E**gység)
DST (**D**efined **S**imbol **T**able)	definiált szimbólumtáblázat
DVH (**D**e**V**ide or **H**alt)	oszt vagy leáll
DVP (**D**e**V**ide or **P**roceed)	oszt vagy folytat
HIO (**H**alt **I/O**)	B/K állj
I/O interface	B/K illesztés
IOD (**I**nput-**O**utput **D**evice)	BKK (**B**e-, ill. **K**imeneti **K**észülék)
IOU (**I**nput- **O**utput **U**nit)	BKE (**B**eviteli-**K**iviteli **E**gység)
IPL (**I**nitial **P**rogram **L**oading)	kezdeti program betöltés
JMP (**JuMP**)	ugrás
JOM (**J**ump **O**n **M**inusz)	ugrás mínuszra
KEY regiszter	KULCS regiszter
L (**L**arge storage)	háttértár

* Az angol és a magyar kifejezéseket a szöveg környezetétől függően igyekeztünk váltakozva használni.

LDQ (**LoaD MQ**)
LLS (**Long Left Shift**)
LRS (**Long Right Shift**)
LS (**Local Storage**)
LSU (**Large Storage Unit**)
MCH (**Multiplexor CHannel**)
MCU (**Microprogram Control Unit**)
MICU (**MIcsoprogrammed Control Unit**)
MPR (**MultiPly and Round**)
MQ (**Multiplier — Quotient**)
MU (**Memory Unit**)
PCI (**Program Controlled Interrupt**)
PRI (**PRogram Interrupt**)
PP (**Peripheral Processor**)
PSW (**Program Status Word**)
RND (**RouND**)
SBM (**SuBtract Magnitude**)
SCH (**Selector CHannel**)
SHR (**SHift Right**)
SIO (**Start I/O**)
SR (**Storage Regiszter**)
STO (**STOre**)
STQ (**STore MQ**)
STP (**STоP**)
SUB (**SUBtraction**)
SVC (**SuperVisor Call**)
TCH (**Test CHannel**)
TIO (**Test IO**)
UCW (**Unit Control Word**)
UST (**Undefined Symbol Table**)

MQ tárolás
hosszú balra léptetés
hosszú jobbra léptetés
HT (Helyi Tár)
HTE (**Háttér Tároló Egység**)
MCS (**Multiplex CSatorna**)
MVE (Mikroprogram Vezérlő Egység)
MIVE (**MIkroprogramozott Vezérlő Egység**)
szorzás és kerekítés
szorzó-hányados
TE (**Tároló Egység**)
program vezérelt megszakítás
program megszakítás
PP (**Perifériás Processzor**)
PSSZ (**Program Statusz SZó**)
kerekítés
LSZ (**Lépés SZámláló**)
SCS (**Selektor CSatorna**)
léptetés jobbra
B/K start
TR (**Tartalom Regiszter**)
tárolás
MQ tárolás
stop
kivonás
felügyelőprogram-hívás
EVSZ (**Egység Vezérlő Szó**)
definiálatlan szimbólumtáblázat
csatorna teszt
B/K teszt

Tárgymutató

A, Á

adatátvitel 377, 433
adatátviteli út szélessége 242
adatcsatorna 412
adatformátumok 369
adatkezelő utasítások 389
adatláncolás 420
adatláncolást jelző bit 419
adatok 377
adatutak 36
akkumulátor regiszter 118
aktivitáslista 280
alapoperátorok 20
állapotbyte 435
általános regiszter 390
argumentum 255
aritmetikai algoritmusok 121
— egység, bináris fixpontos 117, 118
— —, — lebegő pontos 117, 163
— —, — párhuzamos 117, 195
— —, — soros 195
— —, decimális 217
— operátor 20
array-register utasítás 17
— — casregister utasítás 17
ASCII kód 372
aszinkron szervezési elv 340
asszociatív tár 255, 269
áthelyezhető elem 459
— kód 459
— szavak 459
— szóformátumok 460
átlapolás 243
átvitel, bemeneticsoport 140
—, bemenetiszekció 141
—, csoport- 142, 145
—, szekció- 141
— terjedési útja 139
átvitelterjedés 138

B

báziscím 295
báziscímzés 251
BCD kód 371
be-, ill. kimeneti (BK) készülék 367
—, — — — megszakítások 425
—, — — — vezérlés 409
beékelés 243
belső BCD kód 371
bemeneticsoport átvitel 140
bemenetiszekció átvitel 141, 147
betöltő 266
bináris— decimális konverzió 69
— fixpontos aritmetikai egység 117, 118
— kód 65
— lebegőpontos aritmetikai egység 117, 195
— operátor 21
— párhuzamos aritmetikai egység 117, 195
— soros aritmetikai egység 195
— szám formátuma 196
— számok, előjel nélküli 46
— —, előjeles 46
— —, fixpontos 117
— —, lebegőpontos 117
binárisan kódolt decimális kód (BCD) 371
binárisan kódolt decimális számok 217

— — — — ábrázolása 217, 219
— — — — kivonása 218
— — — — összeadása 218
bit 368
bithagyatékonyság 247
bitpárhuzamos keresés 256
B/K illesztés 357, 387, 426
— megszakítás 400
— utasítások 389, 424
— vezérlőegység 367
blokkolt sor 300
blokkutasítás 188
Boole-argumentumra való egyeztetés 265
Booth-féle algoritmus 205
burst-üzemmód 386, 416
byte 368
— — üzemmód

C

casregister 16
— -utasítás 16
CDL 15
célpontbit 254
ciklikus léptetőoperátor 20
ciklusiopás 411
címbyte 435
címezhető egység 247
címfordítási eljárás 292
címke 25
címzés, bázis- 251
—, dinamikus 330
—, direkt 248
—, implicit 252
—, indexelt 249
—, indirekt 249, 276
—, indirekt regiszter- 252
—, regiszter- 251
—, relatív 251
címtér 292
clock-utasítás 19
comment utasítás 28
CPU 366
crossbarkapcsoló 244

CS

csak olvasható tár 381
csatorna 353, 366
— -csatornaadapter 388
—, multiplex- 358, 366, 414
—, szelektor- 366, 414
— vég feltétel 425
csatornaállapot-szó 421
csatornaállapot-szó formátumok 422
csatornacímszó 418
csatornaparancs 412
csatornaparancsszó 418
csatornaprogram 412
csoportátvitel 141, 145
csoportterjedés 141, 146

D

decimális aritmetikai egység 217
— számjegyszorzó 235
decoder-utasítás 19
definiálatlan szimbólumtáblázat 485
deklarációs utasítás 15
dekódoló 19
— -utasítás 19
dinamikus betöltő 266
— címzés 330
— tárallokáció 291
direkt címzés 248
— vezérlés 316

E, É

EBCDIC kód 372
egybites teljes kivonó 197
— — összeadó 197
egyező szavak olvasása művelet 258
egyeztetés határok közötti értékre 261
— kisebb mint értékre 259
— logikai kérdőszóra 265
— maximumra 262
— minimumra 262
— nagyobb mint értékre 259
— numerikus kérdőszóra 258
— számlálási értékre 265
egységvezérlő szó 423
egyszeres elérésű szervezés 245
elhelyezési algoritmus 293
ellenőr 301
előjel nélküli bináris számok 46
előjeles bináris számok 46
emitmező 316
end utasítás 26
érzékelésbyte 435

F

feldolgozó program 367
feltételkód 311, 399
feltételes mikroutasítás 23
felügyelőállapot 397
felülírás 459
figyelemfeltétel 425
fix hosszúságú információ 368
fixpontos aritmetikai egység 117

— bináris számok 117
— szám formátuma 118
fordítási algoritmus 464
formátumok, adat- 369
—, bináris szám 196
—, csatornaállapotszó 397
—, fixpontos szám 118
—, lebegőpontos szám 164
—, parancsszó 419
—, programállapot szó 422
—, utasítás- 328
—, vezérlőszó 92, 105
főtár 380
futásállapot 396
futási sor 300

G

géphiba megszakítás 400
Gray-kód 65
— — konverzió 65

H

háttértároló egység 353
hely 292
helyettesítő algoritmus 293, 300
helyi átvitel 139
— tár 381
hely szerint címezhető tár 255
hexadecimális kód 370
Hollerithkártya kód 374
horizontális mikroprogramozás 316
hosszjelzés elnyomást jelző bit 419

I

időintervallum-adó 405
igényhívás 299
implicit címzés 252
indexelt címzés 249
indikátor utasítás 18
indirekt címzés 249, 276
— regisztercímzés 252
információcsere amerikai standard kód (ASCII) 372
írj a következő helyre művelet 258

J

jelölj meg minden bitet művelet 258

K

kapcsolóutasítás 18
kapocsvár-utasítás 19, 142
kapu 244
kapuzás 319
karakterisztika 164
karakterisztikák azonos értékűvé tétele 170, 187
készenléti sor 300
készülék vég feltétel 425
készülékvég típusú megszakítás 426
kétlépcsős regiszter 148
kezdeti-program betöltés 406
— kiválasztás 437
kiegészítő bit 380
kifejezés 21
kimaradt szegmens hibajelzés 295
kimeneti kapocspár 244
kiterjesztett binárisan kódolt decimális cserekód (EBCDIC) 372
kód, áthelyezhető 459
—, belső BCD 371
—, bináris 65
—, binárisan kódolt decimális 371
—, feltétel 311, 399
—, hexadecimális 370
—, Hollerithkártya 374
—, információcsere amerikai standard 372
—, kiterjesztett binárisan kódolt decimális csere- 372
—, külső BCD 372
—, megszakítási 399
—, önkomplementáló 218
—, utasításhossz- 375, 399
közös szimbólumtáblázat 485
központi feldolgozó egység 366
közvetlen elérésű tár 240, 245
külső BCD kód 371
— megszakítás

L, Ly

lap 344
lapcím 268
lapok 293
laptábla 293
lapvárakozási sor 300
last-in-first-out lista 252
lebegőpontos alulcsordulás 165
— aritmetikai egység 163
— bináris számok 117
— regiszter 390
lebegőpontos szám formátuma 164
— túlcsordulás 165
lehívási algoritmus 293
lekérdezés 431
lépésszámláló regiszter 118
léptetőoperátor 20
light-utasítás 18

logikai operátor 20
logout 385

M

magyarázó utasítás 28
megszakítás-felügyelőprogram 403
megszakítási kód 399
— rendszer 333
megszakítható állapot 396
memory-utasítás 17
Mercer-algoritmus 74
mikroművelet 20, 22, 35, 101
mikroprogram 89, 403
— aritmetikai egységhez 160
— betöltőhöz 486
— legnagyobb szám meghatározásához 114
— mikroprogramozott számítógéphez 101
— paritásgeneráló hálózathoz 91
— tárolt logikájú számítógéphez 108
mikroprogramozás 89
mikroprogramozott B/K (I/O) vezérlő egység 324
— CPU 31
— számítógép 89
— vezérlés 89
— — struktúrája CPU-hoz 318
— — — egy sorozatból a legnagyobb szám meghatározására 109
— — — kétszintes hierarchiához 315
— — — MVE-hez 325
— — — párhuzamos bináris fixpontos aritmetikai egység realizálásához 152
— — — soros paritást ellátó hálózathoz 90
— — — tárolt logikájú számítógéphez 103
— — — tárolt programú számítógéphez 95
— — — töltőhöz 457
— — — vezérlőegységhez 312
— -vezérlőegység 311
mikroprogramtár 89
mikroszubrutin 403
mikroutasítás 22, 91
—, feltételes 23
modul 243
moduláris felépítés 359
modulszervezés 242
multiplex üzemmód 386, 415
multiplexcsatorna 358, 366, 414
multiplexelés 358
multiplextár 380
munka 458
működésállapot 32, 396
működtető rendszer 367

N, Ny

negatív nulla 18
nemegyező szavak olvasása művelet 258
— — számlálása művelet 358
nem szimmetrikus egyezés 265
név 292
normál alak 165
— nulla 166
— sorrend 32
normalizálás 166, 172

O, Ö

olvass utasítás
operátor, alap- 20
—, aritmetika 20
—, bináris 21
—, ciklikus léptető- 20
—, léptető 20
—, logikai 20
—, regiszter- 20
—, speciális 24
—, unáris 21
órajelciklus 19
osztási túlcsordulás 129, 208
osztott ciklusú műveletre alkalmas tár 242
— olvasás—írás ciklus 380
önkomplementáló kód 218
összeadási túlcsordulás 123, 202
összekapcsoló mikroműveletek 319

P

parancsbyte 435
parancsláncolás 420
parancsolást jelző bit 419
parancsszó formátumok 419
páratlan paritásbit 42
párhuzamos aritmetikai egység 117, 195
— decimális összeadó 220
— keresésű tár 256
— összeadó 18, 120
— számjegy üzemmód 220
páros paritásbit 42
pozitív nulla
primszámgenerátor 59
privilegizált utasítás 397
programállapotszó 397
— formátumok 397
programhiba megszakítás 400
programmaszk 399
programvezérelt megszakítás feltétel 426
— megszakítást jelző bit 419
PS átkapcsolás 402

—, light- 18
—, magyarázó 28
—, memory- 17
—, olvass 409
—, privilegizált 397
—, register- 16
—, regisztertömb 17
—, rendszervezérlő 389
—, subregister- 16
—, tár- 17
— tárolj 409
—, tárolj 409
—, terminál 19, 142
—, végrehajtási 25
utasításcím 399
utasítás formátumok 328
utasításhosszkód 375, 399
ütemezési eljárás 300

V

változó hosszúságú információ 368
várakozási állapot 32, 396
védelmi kulcs 404
végrehajtási utasítás 25
— ciklus 134
végrehajtható kódú program 463
vég-típusú megszakítás 426
vertikális mikroprogramozás 316
vezérlési ciklus 32, 93
— utak 36
vezérlőegység vég feltétel 425
vezérlőprogram 367
vezérlőszavak 91, 311
vezérlőszó formátumok 92, 108
virtuális cím 292
— tár 291
visszaállítás nélküli algoritmus 208

Z

zónázott decimális formátum 370, 374

Műszaki vezető: Hegedűs Ernő — Műszaki szerkesztő: Molnár József
Borító- és kötésterv: Fábri Péter
A könyv ábráit rajzolta: Sándor Tibor
A könyv formátuma: B5 — Ívterjedelem: 43,375 (A5)
Ábrák száma: 214 — Példányszám: 4400
Papír minősége: 80 g ofszet — Betűcsalád és méret: gm/gm Gill monó
Azonossági szám: 60769 — MŰ: 2546-h-7780
Készült az MSZ 5601—59 és 5602—67 szerint
77-1995 Szegedi Nyomda. Felelős vezető: Dobó József igazgató